OpenVMS User's Manual

December 1995

This manual describes how to use the OpenVMS operating system. The information contained in this manual is intended for all OpenVMS users and is applicable to all computers running the OpenVMS operating system.

Revision/Update Information:	This manual supersedes the *OpenVMS User's Manual*, OpenVMS Alpha Version 6.2 and OpenVMS VAX Version 6.2.
Software Version:	OpenVMS Alpha Version 7.0 OpenVMS VAX Version 7.0

Digital Equipment Corporation
Maynard, Massachusetts

December 1995

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

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

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

© Digital Equipment Corporation 1995. All rights reserved.

The following are trademarks of Digital Equipment Corporation: Bookreader, CDA, CDD/Repository, DEC Ada, DEC C, DECdns, DECdtm, DEC Fortran, DECnet, DECprint, DECram, DECspell, DECUS, DECwindows, DECwriter, DEQNA, Desktop–VMS, DEUNA, Digital, EDT, GIGI, HSC, LiveLink, LN03, MASSBUSS, MicroVAX, OpenVMS, OpenVMS RMS, PrintServer 40, Q–bus, ReGIS, RSX, ULTRIX, UNIBUS, VAX, VAX BASIC, VAXBI, VAXcluster, VAX COBOL, VAX DOCUMENT, VAX FORTRAN, VAX MACRO, VAXserver, VAXstation, VMS, VMScluster, VT, VT100, VT220, VT300, WPS, WPS–PLUS, XUI, and the DIGITAL logo.

The following are third-party trademarks:

Adobe, Adobe Illustrator, Display PostScript, and PostScript are registered trademarks of Adobe Systems Incorporated.

BASIC is a registered trademark of Trustees of Dartmouth College, D.B.A. Dartmouth College.

Motif is a registered trademarks of the Open Software Foundation, Inc.

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

ZK4323

This document is available on CD–ROM.

Contents

Preface xli

1 Introduction: OpenVMS Concepts and Definitions

1.1 Overview 1–1
1.1.1 Overview of OpenVMS 1–1
1.1.2 Differences in Your Local Environment 1–1
1.1.3 References 1–2
1.2 Logging In to the System 1–2
1.2.1 Accounts 1–2
1.2.2 Access Requirements 1–2
1.2.3 Logging In 1–2
1.3 Networks 1–3
1.3.1 Overview 1–3
1.3.2 Network Nodes 1–3
1.3.3 Executing Programs Over Networks 1–3
1.3.4 Task-to-Task Communication 1–3
1.3.5 Proxy Acounts 1–3
1.4 DIGITAL Command Language (DCL) 1–4
1.4.1 Overview 1–4
1.4.2 Usage Modes 1–4
1.4.3 Types of DCL Commands 1–4
1.4.4 DCL Command Line 1–5
1.4.5 DCL Command Line Format 1–5
1.4.6 Lexical Functions 1–5
1.5 Files and Directories 1–5
1.5.1 Definition: File 1–5
1.5.2 Definition: Directory 1–5
1.5.3 Hardware 1–5
1.5.4 File Specifications 1–6
1.5.5 Directory Structures 1–6
1.5.6 Subdirectories 1–6
1.6 OpenVMS Utilities 1–7
1.6.1 The Mail Utility 1–7
1.6.2 The OpenVMS Phone Utility 1–7
1.6.3 Text Editors 1–7
1.6.4 DIGITAL Standard Runoff (DSR) 1–7
1.6.5 The Sort/Merge Utility 1–7
1.7 Devices 1–8
1.7.1 Mass Storage Devices 1–8
1.7.2 Record Oriented Devices 1–8
1.7.3 Disks and Magnetic Tapes 1–8
1.7.4 For Additional Information 1–8
1.8 Logical Names 1–8

1.8.1 Overview . . . 1–8
1.8.2 Readability . . . 1–8
1.8.3 File Independence . . . 1–9
1.8.4 For Additional Information . . . 1–9
1.9 Symbols . . . 1–9
1.9.1 Definition: Symbols . . . 1–9
1.9.2 When to Use Symbols . . . 1–9
1.9.3 For Additional Information . . . 1–9
1.10 Command Procedures . . . 1–9
1.10.1 Definition: Command Procedure . . . 1–9
1.10.2 System Login Command Procedures . . . 1–9
1.10.3 Personal Login Command Procedures . . . 1–10
1.10.4 Creating Login Command Procedures . . . 1–10
1.10.5 For Additional Information . . . 1–10
1.11 Lexical Functions . . . 1–10
1.11.1 Definition: Lexical Functions . . . 1–10
1.11.2 Usage . . . 1–10
1.11.3 For Additional Information . . . 1–10
1.12 Processes and Programs . . . 1–11
1.12.1 User Authorization Files (UAFs) . . . 1–11
1.12.2 Processes . . . 1–11
1.12.3 Programs . . . 1–11
1.12.4 Creating Image Files . . . 1–11
1.13 System Security . . . 1–12
1.13.1 Overview . . . 1–12
1.13.2 Protected Objects . . . 1–12
1.13.3 For Additional Information . . . 1–12

2 Getting Started: Interacting with the OpenVMS Operating System

2.1 Overview . . . 2–1
2.1.1 Site-Specific Procedures . . . 2–1
2.1.2 References . . . 2–1
2.2 Logging In to the System . . . 2–2
2.2.1 Introduction . . . 2–2
2.2.2 How to Log In . . . 2–2
2.2.3 Successful Logins . . . 2–2
2.2.4 Example . . . 2–2
2.2.5 Login Errors . . . 2–3
2.3 Choosing Passwords for Your Account . . . 2–3
2.3.1 Guidelines . . . 2–3
2.3.2 Secure and High-Risk Passwords . . . 2–3
2.3.3 Obtaining Your Initial Password . . . 2–4
2.3.4 Changing Your Initial Password . . . 2–4
2.3.5 Restrictions on Passwords . . . 2–4
2.3.6 Types of Passwords . . . 2–4
2.3.7 Entering a System Password . . . 2–5
2.3.8 How to Enter a System Password . . . 2–5
2.3.9 Entering a Secondary Password . . . 2–6
2.3.10 Example . . . 2–6
2.3.11 Password Requirements for Different Types of Accounts . . . 2–6
2.4 Reading Informational Messages . . . 2–7
2.4.1 Introduction . . . 2–7
2.4.2 Example: Local Login Messages . . . 2–7

2.4.3 Suppressing Messages 2–8
2.4.4 Successful Login Messages 2–9
2.5 Types of Logins and Login Classes 2–9
2.5.1 Overview 2–9
2.5.2 Login Classes 2–9
2.5.3 Interactive Logins 2–9
2.5.4 Noninteractive Logins 2–10
2.6 Login Failures 2–10
2.6.1 Common Login Failures 2–10
2.6.2 Terminals That Require System Passwords 2–11
2.6.3 Login Class Restrictions 2–11
2.6.4 Shift Restrictions 2–12
2.6.5 Batch Jobs During Shift Restrictions 2–12
2.6.6 Failures During Dialup Logins 2–12
2.6.7 Break-In Evasion Procedures 2–12
2.7 Changing Passwords 2–13
2.7.1 Overview 2–13
2.7.2 Example: Password Error Messages 2–13
2.7.3 Selecting Your Own Password 2–13
2.7.4 Using Generated Passwords 2–14
2.7.5 Example 2–14
2.7.6 Generated Passwords: Disadvantages 2–15
2.7.7 Changing a Secondary Password 2–15
2.7.8 Changing Passwords at Login 2–16
2.7.9 Example 2–16
2.8 Password and Account Expiration Times 2–16
2.8.1 Overview 2–16
2.8.2 Expired Passwords 2–16
2.8.3 Using Secondary Passwords 2–17
2.8.4 Failure to Change Passwords 2–17
2.8.5 Expired Accounts 2–17
2.9 Guidelines for Protecting Your Password 2–17
2.9.1 Overview 2–17
2.9.2 Guidelines 2–17
2.10 Recognizing System Responses 2–19
2.10.1 Overview 2–19
2.10.2 Default Actions 2–19
2.10.3 Informational System Messages 2–20
2.10.4 System Error Messages 2–20
2.10.5 System Error Messages During Command Execution 2–20
2.10.6 Checking Your Current Process 2–21
2.10.7 Enabling Ctrl/T 2–21
2.11 Getting Help About the System 2–21
2.11.1 Overview 2–21
2.11.2 Using Online Help 2–22
2.11.3 Example 2–22
2.11.4 Getting Help on Specific Commands 2–23
2.11.5 Getting Help on System Messages 2–23
2.12 Logging Out of the System 2–24
2.12.1 Overview 2–24
2.12.2 LOGOUT Command 2–24
2.12.3 When to Log Out 2–24
2.12.4 Obtaining Accounting Information 2–24
2.12.5 Ending a Remote Session 2–24

2.12.6 Lost Network Connections 2–25
2.13 Logging Out Without Compromising System Security 2–25
2.13.1 Protecting Your Resources 2–25
2.13.2 Reasons to Clear Your Terminal Screen 2–25
2.13.3 Clearing Your Terminal Screen 2–25
2.13.4 Disposing of Hardcopy Output 2–26
2.13.5 Breaking the Connection to a Dialup Line 2–26
2.13.6 Reasons to Break the Connection to Dialup Lines 2–26

3 The DIGITAL Command Language: Interacting with the System

3.1 Overview 3–1
3.1.1 References 3–1
3.2 Using DCL Commands 3–1
3.2.1 Definition: DIGITAL Command Language (DCL) 3–1
3.2.2 Entering Commands 3–2
3.2.3 Example 3–2
3.2.4 Commonly Used DCL Commands 3–2
3.2.5 Key Sequences 3–2
3.2.6 Commonly Used Key Sequences 3–3
3.3 Constructing DCL Commands 3–3
3.3.1 Overview 3–3
3.3.2 Format of a DCL Command 3–3
3.3.3 Other Components of Command Lines 3–4
3.3.4 Syntax 3–4
3.4 Entering DCL Commands 3–5
3.4.1 Specifying Parameters 3–5
3.4.2 Example 3–5
3.4.3 Cancelling Commands 3–5
3.4.4 Using Defaults 3–5
3.4.5 Example 3–5
3.4.6 Entering Multiple Line Commands 3–6
3.4.7 Example 3–6
3.5 Rules for Entering DCL Commands 3–6
3.5.1 Case Sensitivity 3–6
3.5.2 Required Spaces 3–6
3.5.3 Required Punctuation 3–7
3.5.4 Null Characters 3–7
3.5.5 Maximum Elements 3–7
3.5.6 Abbreviating Commands 3–7
3.5.7 Example 3–7
3.5.8 Commands in Command Procedures 3–7
3.6 Entering Parameters 3–7
3.6.1 Common Parameters 3–7
3.6.2 Rules for Specifying Parameters 3–8
3.6.3 Examples 3–8
3.7 Entering Qualifiers 3–8
3.7.1 Types of Qualifiers 3–8
3.7.2 Abbreviating Qualifiers 3–9
3.7.3 Default Qualifiers 3–9
3.7.4 Command Qualifiers 3–9
3.7.5 Example 3–9
3.7.6 Positional Qualifiers 3–9
3.7.7 Example 3–9

3.7.8 Parameter Qualifiers . . . 3–9
3.7.9 Example . . . 3–10
3.7.10 Conflicting Qualifiers . . . 3–10
3.7.11 Example . . . 3–10
3.7.12 Values Accepted by Qualifiers . . . 3–10
3.7.13 Examples . . . 3–10
3.8 Entering Dates and Times as Values . . . 3–11
3.8.1 Overview . . . 3–11
3.8.2 Absolute Time Format . . . 3–11
3.8.3 Rules for Specifying Absolute Time Format . . . 3–11
3.8.4 Absolute Time Keywords . . . 3–12
3.8.5 Examples: Absolute Time . . . 3–12
3.8.6 Delta Time Format . . . 3–12
3.8.7 Rules for Specifying Delta Time Format . . . 3–12
3.8.8 Examples . . . 3–13
3.8.9 Combination Time Format . . . 3–13
3.8.10 Rules for Specifying Combination Time . . . 3–13
3.8.11 Examples . . . 3–13
3.9 Recalling Commands . . . 3–14
3.9.1 Overview . . . 3–14
3.9.2 Using Ctrl/B . . . 3–14
3.9.3 Using Arrow Keys . . . 3–14
3.9.4 Using the RECALL Command . . . 3–15
3.9.5 Examples . . . 3–15
3.9.6 OpenVMS Screen Management Software . . . 3–15
3.9.7 Erasing the Recall Buffer . . . 3–15
3.10 Editing the DCL Command Line . . . 3–16
3.10.1 Overview . . . 3–16
3.10.2 SHOW TERMINAL Command . . . 3–16
3.10.3 Example . . . 3–16
3.10.4 SET TERMINAL Command . . . 3–16
3.10.5 Enabling Line Editing . . . 3–16
3.10.6 Changing Edit Modes . . . 3–16
3.10.7 Making Command Lines Wrap . . . 3–17
3.10.8 Deleting Portions of the Command Line . . . 3–17
3.11 Defining Terminal Keys . . . 3–17
3.11.1 Key Definitions . . . 3–17
3.11.2 Definable Keys . . . 3–17
3.12 Summary of Key Sequences . . . 3–18
3.12.1 Keys That Enter DCL Commands . . . 3–18
3.12.2 Keys That Interrupt DCL Commands . . . 3–18
3.12.3 Keys That Recall Commands . . . 3–19
3.12.4 Keys That Control Cursor Position . . . 3–19
3.12.5 Keys That Control Screen Display . . . 3–20

4 Files: Storing Information

4.1 Overview . . . 4–1
4.1.1 References . . . 4–1
4.2 Understanding File Names and File Specifications . . . 4–1
4.2.1 Overview . . . 4–1
4.2.2 Providing a Complete File Specification . . . 4–2
4.2.3 Rules for File Specifications . . . 4–2
4.2.4 Default File Types Used by DCL Commands . . . 4–3

4.2.5 Default File Types for Language Source Programs 4–4
4.2.6 File Versions 4–5
4.2.7 Specifying File Versions 4–5
4.2.8 Controlling the Number of File Versions 4–5
4.2.9 Network Node Names 4–6
4.2.10 Node Specification Format and Rules 4–6
4.2.11 Specifying Node Full Names 4–6
4.2.12 Examples 4–7
4.2.13 Accessing Files on Remote Nodes 4–7
4.2.14 Note: Examples in This Chapter 4–7
4.2.15 Using Network File Specifications 4–7
4.2.16 File Name Format 4–8
4.2.17 Foreign File Specification 4–8
4.2.18 Example 4–8
4.2.19 Task Specification Strings 4–8
4.2.20 Example 4–8
4.2.21 Note: ULTRIX Restrictions 4–8
4.2.22 Access Control String Format 4–8
4.2.23 Example 4–9
4.3 Using Wildcards with File Names 4–9
4.3.1 Overview 4–9
4.3.2 Types of Wildcards 4–9
4.3.3 The Asterisk (*) Wildcard Character 4–9
4.3.4 Examples 4–9
4.3.5 The Percent Sign Wildcard Character 4–10
4.3.6 Examples 4–10
4.4 Other File Names 4–10
4.4.1 Null File Names and File Types 4–10
4.4.2 Example 4–11
4.4.3 Alternate File Names for Magnetic Tapes 4–11
4.5 Creating and Modifying Files 4–11
4.5.1 Text Editors 4–11
4.5.2 Commands That Create Files 4–11
4.5.3 Creating Files 4–11
4.5.4 Example 4–11
4.5.5 Copying Files 4–12
4.5.6 Examples 4–12
4.5.7 File Concatenation 4–12
4.5.8 Copying Files from a Remote Node to Your Node 4–12
4.5.9 Copying Files from Your Node to a Remote Node 4–12
4.5.10 Example 4–12
4.5.11 Using Mail to Copy Files 4–13
4.5.12 Example 4–13
4.5.13 Using Access Control Strings to Copy Files 4–13
4.5.14 Example 4–13
4.5.15 Renaming Files 4–13
4.5.16 Example 4–13
4.6 Displaying the Contents of Files 4–14
4.6.1 Using the TYPE Command 4–14
4.6.2 Example 4–14
4.6.3 Controlling the Display 4–14
4.6.4 Using Text Editors to Display Files 4–14
4.6.5 Displaying Files on Remote Nodes 4–14
4.6.6 Example 4–14

4.6.7 Displaying Files with Wildcards 4–14
4.6.8 Examples 4–14
4.6.9 Displaying Multiple Files 4–15
4.7 Deleting Files 4–15
4.7.1 Using the DELETE Command 4–15
4.7.2 Examples 4–15
4.7.3 DELETE Command Qualifiers 4–15
4.7.4 Examples 4–15
4.7.5 Using the PURGE Command 4–16
4.7.6 Example 4–16
4.8 Protecting Files from Other Users 4–16
4.8.1 Access Control Lists (ACLs) 4–16
4.8.2 Types of Protection 4–16
4.8.3 For Additional Information 4–16
4.9 Printing Files 4–16
4.9.1 Using the PRINT Command 4–16
4.9.2 Example 4–17
4.9.3 Print Job Priority 4–17
4.9.4 Displaying Queue Information 4–17
4.9.5 Example 4–18
4.9.6 Print Forms 4–18
4.9.7 Stopping a Print Job 4–18
4.9.8 Example 4–18
4.9.9 Printing Files on Other Nodes 4–18
4.9.10 Example 4–18
4.9.11 Using the /REMOTE Qualifier 4–18
4.9.12 Print Command Qualifiers 4–19
4.9.13 Summary of Commands That Control Print Jobs 4–19

5 Directories: Organizing and Managing Files

5.1 Overview 5–1
5.1.1 Note: Examples Used in This Chapter 5–1
5.1.2 References 5–1
5.2 Directory Structures 5–1
5.2.1 Example of a Directory Structure 5–1
5.3 Understanding Directories 5–3
5.3.1 Directory Specifications 5–3
5.3.2 Directory Specification Format 5–3
5.3.3 Creating Directories 5–3
5.3.4 Examples 5–3
5.3.5 Displaying Directories 5–3
5.3.6 Examples 5–4
5.3.7 Deleting Directories 5–4
5.3.8 Example 5–5
5.4 Defaults 5–5
5.4.1 Changing Your Default Directory 5–5
5.4.2 Examples 5–5
5.4.3 Setting Default to Nonexistant Directories 5–6
5.4.4 SHOW DEFAULT Command 5–6
5.4.5 Setting Default Devices 5–6
5.4.6 Examples 5–6
5.4.7 Using Temporary Defaults 5–6
5.4.8 Examples 5–7

5.5 Protecting Directories from Other Users 5–7
5.5.1 Reasons to Protect Directories 5–7
5.5.2 Private Files 5–7
5.5.3 Default Directory Protection 5–7
5.5.4 UIC-Based Protection 5–7
5.5.5 Limiting Directory Access 5–8
5.6 Using Wildcards to Search the Directory Structure 5–8
5.6.1 Overview 5–8
5.6.2 Ellipsis Wildcard Character 5–8
5.6.3 Examples 5–8
5.6.4 Hyphen Wildcard Character 5–9
5.6.5 Examples 5–9
5.7 Working with Directories in UIC Format 5–10
5.7.1 Overview 5–10
5.7.2 UIC Directory Format and Rules 5–10
5.7.3 Using Wildcards with UIC Directories 5–10
5.7.4 Translating to Named from UIC Format 5–10
5.7.5 Examples 5–10

6 Mail: Communicating with Other Users

6.1 Overview 6–1
6.1.1 References 6–1
6.1.2 Figure: Sample Mail Message 6–2
6.2 Invoking and Exiting Mail 6–2
6.2.1 Invoking Mail 6–2
6.2.2 Using Mail 6–2
6.2.3 Exiting from Mail 6–3
6.3 Reading Messages 6–3
6.3.1 Overview 6–3
6.3.2 New Mail Notification 6–3
6.3.3 Reading New Mail 6–3
6.3.4 Reading More Than One Message 6–3
6.3.5 Reading New Mail While in Mail 6–3
6.3.6 Reading Old Messages 6–4
6.3.7 Reading Specific Old Mail 6–4
6.3.8 Example 6–4
6.3.9 Searching for Messages 6–4
6.3.10 Example 6–5
6.4 Sending Messages 6–5
6.4.1 How to Send Mail 6–5
6.4.2 Example 6–5
6.5 Sending Mail Over Networks 6–6
6.5.1 Specifying Node Names 6–6
6.5.2 Example 6–6
6.5.3 Using Logical Node Names 6–6
6.5.4 Example 6–6
6.6 Sending Messages to Multiple Users 6–6
6.6.1 Using Individual Names 6–6
6.6.2 Example 6–6
6.6.3 Distribution Lists 6–7
6.6.4 Default File Type for Distribution Lists 6–7
6.6.5 How to Create Distribution Lists 6–7
6.6.6 Example 6–7

6.6.7 Sending Messages to Distribution Lists 6–8
6.6.8 Example 6–8
6.6.9 Sending Mail to Distribution Lists from DCL 6–8
6.6.10 Examples 6–8
6.7 Manipulating Files in Mail 6–9
6.7.1 How to Send Files 6–9
6.7.2 Example 6–9
6.7.3 Sending DDIF Files 6–9
6.7.4 Sending Files from DCL 6–9
6.7.5 Examples 6–10
6.7.6 Creating Files from Messages 6–10
6.7.7 Examples 6–10
6.7.8 Appending Files to Messages 6–11
6.7.9 Examples 6–11
6.8 Other Ways to Send Messages 6–12
6.8.1 Replying to Messages 6–12
6.8.2 Example 6–12
6.8.3 Forwarding Messages 6–12
6.8.4 Example 6–12
6.9 Organizing Messages 6–12
6.9.1 Folders 6–12
6.9.2 Creating Mail Subdirectories 6–13
6.9.3 Example 6–13
6.9.4 Moving Messages into Folders 6–13
6.9.5 Copying Messages 6–13
6.9.6 Example 6–14
6.9.7 Selecting Folders 6–14
6.9.8 Example 6–14
6.9.9 Deleting Folders 6–15
6.9.10 Example 6–15
6.9.11 Creating and Accessing Mail Files 6–15
6.9.12 Figure: Organization of a Typical Mail Directory 6–15
6.9.13 Examples: Creating and Accessing Mail Files 6–15
6.9.14 Correcting the Mail Message Count 6–16
6.10 Deleting Messages 6–16
6.10.1 Using the DELETE Command 6–16
6.10.2 Example 6–16
6.10.3 Recovering Deleted Messages 6–16
6.10.4 Example 6–17
6.11 Printing Mail Messages 6–17
6.11.1 Using the PRINT Command 6–17
6.11.2 Specifying Print Queues 6–17
6.11.3 Examples 6–17
6.12 Protecting Mail Files 6–17
6.12.1 Default Protection 6–17
6.12.2 Security Measures 6–18
6.13 Using Text Editors in Mail 6–18
6.13.1 Using EVE 6–18
6.13.2 Example 6–18
6.13.3 Note: DDIF Files 6–18
6.13.4 Using /EDIT Qualifier Keywords 6–18
6.13.5 Examples 6–19
6.13.6 Default Editor 6–19
6.13.7 Selecting an Editor 6–19

6.13.8 Displaying the Selected Editor 6–19
6.13.9 Example 6–19
6.13.10 Using the EDT Editor 6–20
6.13.11 Using a Command File to Edit Mail 6–20
6.13.12 Overriding Your Selected Editor 6–20
6.14 Customizing Your Mail Environment 6–21
6.14.1 Using the Mail Keypad 6–21
6.14.2 Mail Utility Keypad 6–21
6.14.3 Example of Using the Keypad 6–21
6.14.4 Redefining Keypad Keys 6–21
6.14.5 Example 6–22
6.14.6 Assigning Additional Definitions 6–22
6.14.7 Example 6–22
6.14.8 Creating Permanent Key Definitions 6–22
6.15 Summary of Mail Commands 6–23
6.15.1 Overview 6–23
6.15.2 Reading Messages 6–23
6.15.3 Exchanging Messages 6–24
6.15.4 Removing Messages 6–24
6.15.5 Printing Messages 6–25
6.15.6 Organizing Messages 6–26
6.15.7 Marking Messages 6–27
6.15.8 Customizing the Mail Environment 6–27
6.15.9 Exiting or Transferring Control 6–30
6.15.10 Mail File Compression 6–30
6.15.11 System Management Commands 6–31

7 Phone: Communicating with Other Users

7.1 Overview 7–1
7.1.1 References 7–1
7.2 Using Phone 7–1
7.2.1 Phone Features 7–1
7.2.2 Invoking Phone 7–1
7.2.3 Example of a Phone Viewport 7–2
7.2.4 Help on Phone 7–2
7.2.5 Viewport 7–2
7.3 Entering Phone Commands 7–3
7.3.1 Switchhook 7–3
7.3.2 Refreshing the Screen 7–3
7.3.3 Logical Names 7–3
7.3.4 Special Characters 7–3
7.4 Customizing Your Phone Viewport 7–3
7.4.1 Using PHONE Command Qualifiers 7–3
7.4.2 For Additional Information 7–3
7.5 Summary of Phone Commands 7–4
7.5.1 Phone Commands 7–4
7.5.2 For Additional Information 7–4

8 Editing Text Files: Using EVE

8.1 Overview 8–1
8.1.1 Conventions 8–1
8.1.2 References 8–2
8.2 EVE Features 8–2
8.2.1 Overview 8–2
8.2.2 EVE Usage 8–2
8.2.3 Modifying Files 8–3
8.3 Getting Help 8–3
8.3.1 Using Keypad Help 8–3
8.3.2 Using EVE Help 8–3
8.3.3 Example 8–4
8.4 Beginning an Editing Session 8–4
8.4.1 Invoking EVE 8–4
8.4.2 Example: Invoking EVE 8–4
8.5 Entering Commands 8–5
8.5.1 Overview 8–5
8.5.2 Typing Commands 8–5
8.5.3 Using Defined Keys 8–6
8.5.4 Figure: EVE Keys 8–6
8.5.5 Figure: EVE Keys—VT100 Series Terminals 8–7
8.6 Saving Your Edits and Exiting from EVE 8–7
8.6.1 Overview 8–7
8.6.2 Using the WRITE FILE Command 8–8
8.6.3 Using the EXIT Command 8–8
8.6.4 Using the QUIT Command 8–8
8.6.5 Example 8–8
8.7 Moving the Cursor 8–8
8.7.1 Overview 8–8
8.7.2 EVE Editing Keys That Move the Cursor 8–9
8.7.3 EVE Commands That Move the Cursor 8–10
8.7.4 Tutorial: Moving the Cursor in EVE 8–12
8.8 Entering Text 8–13
8.8.1 Overview 8–13
8.8.2 Adding Text 8–13
8.8.3 Including Files 8–13
8.8.4 Special Nonprinting Characters 8–13
8.8.5 EVE Editing Keys for Entering Text 8–13
8.8.6 EVE Commands for Entering Text 8–14
8.8.7 Setting Buffer Mode 8–14
8.8.8 Tutorial: Adding Text 8–15
8.9 Erasing and Restoring Text 8–16
8.9.1 Overview 8–16
8.9.2 EVE Editing Keys for Erasing and Restoring Text 8–16
8.9.3 EVE Commands for Erasing and Restoring Text 8–17
8.9.4 Tutorial: Erasing and Restoring Text 8–18
8.10 Moving Text 8–19
8.10.1 Overview 8–19
8.10.2 How to Move Text 8–19
8.10.3 EVE Editing Keys That Move Text 8–20
8.10.4 EVE Commands That Move Text 8–21
8.10.5 Tutorial: Moving Text 8–22
8.11 Copying Text 8–22
8.11.1 Overview 8–22

8.11.2 Tutorial: Copying Text 8–23
8.12 Box Editing 8–23
8.12.1 Overview 8–23
8.12.2 Selecting a Box of Text 8–23
8.12.3 Cutting and Pasting a Box of Text 8–24
8.12.4 EVE Commands for Box Editing 8–24
8.12.5 Tutorial: Cutting and Pasting Text 8–25
8.12.6 SET BOX SELECT Commands 8–25
8.13 Using Pending Delete 8–26
8.13.1 Overview 8–26
8.13.2 Erasing a Selection with Pending Delete 8–26
8.13.3 Restoring a Selection That Was Erased with Pending Delete 8–26
8.13.4 Effects on Box Editing 8–26
8.14 Finding and Replacing Text 8–27
8.14.1 Overview 8–27
8.14.2 EVE Commands for Locating Text in a Buffer 8–27
8.14.3 Finding Text 8–28
8.14.4 Search Direction 8–28
8.14.5 When a Search String is Found 8–28
8.14.6 Setting Case-Exact Searches 8–29
8.14.7 Example 8–29
8.14.8 Tutorial: Finding Text 8–29
8.14.9 Tutorial: Using the FIND SELECTED Command 8–30
8.14.10 Using Wildcards 8–30
8.14.11 Tutorial: Using Wildcards 8–30
8.14.12 Including White Space in a Search 8–30
8.14.13 Marking Locations in Text 8–31
8.14.14 Replacing Text 8–31
8.14.15 REPLACE Command and Case Sensitivity 8–31
8.14.16 Case Handling by EVE 8–32
8.14.17 SET FIND CASE EXACT Command 8–32
8.14.18 REPLACE Command Responses 8–32
8.15 Using Command Line Qualifiers 8–33
8.15.1 Overview 8–33
8.15.2 EDIT/TPU Command Line Qualifiers 8–33
8.15.3 Starting in an Alternate Position 8–33
8.15.4 /START_POSITION Qualifier Format 8–34
8.15.5 Using the /START_POSITION Qualifier 8–34
8.15.6 Work Files 8–34
8.15.7 Specifying Work Files 8–34
8.15.8 Modifying the Main Buffer 8–35
8.15.9 Overriding the /READ_ONLY Qualifier 8–35
8.16 Alternate Methods to Invoke EVE 8–35
8.16.1 Overview 8–35
8.16.2 Example: Invoking EVE from a Search List 8–35
8.16.3 Invoking EVE with Wildcards 8–36
8.16.4 Examples 8–36
8.16.5 Invoking EVE with Wildcard Directory Names 8–36
8.16.6 Example 8–36
8.16.7 Invoking EVE with Multiple Input Files 8–37
8.17 Journaling and Recovery 8–37
8.17.1 Overview 8–37
8.17.2 Disabling Journaling 8–37
8.17.3 Running File Backups 8–37

8.17.4 Using Buffer-Change Journaling 8–37
8.17.5 EVE Commands for Buffer Change Journaling and Recovery 8–38
8.17.6 Buffer Change Journal Files 8–38
8.17.7 Deriving Buffer Change Journal Names 8–38
8.17.8 Buffer Change Journal File Names 8–39
8.17.9 Using Buffer-Change Journaling to Recover Edits 8–39
8.17.10 Using the /RECOVER Qualifier 8–39
8.17.11 Using the RECOVER BUFFER Command 8–39
8.17.12 How to Recover When You Are Unsure of the File Name 8–40
8.17.13 How to Recover All Buffers 8–40
8.17.14 Disabling Buffer-Change Journaling 8–40
8.17.15 Enabling Buffer-Change Journaling 8–41
8.18 EVE Formatting Commands 8–41
8.18.1 Overview 8–41
8.18.2 EVE Editing Keys and Their Functions 8–41
8.18.3 EVE Text Formatting Commands and Their Functions 8–42
8.19 Using Buffers 8–44
8.19.1 Definition: Buffers 8–44
8.19.2 EVE Commands to Manipulate Buffers 8–44
8.19.3 Obtaining Buffer Information 8–46
8.19.4 Deleting a Buffer 8–46
8.19.5 Example 8–47
8.19.6 Changing Buffer Status 8–47
8.19.7 SET BUFFER Command Keywords 8–47
8.19.8 Changing Buffer Status 8–47
8.19.9 Displaying the Messages Buffer 8–48
8.19.10 Editing Multiple Buffers 8–48
8.19.11 Using the GET FILE Command 8–48
8.19.12 Using the OPEN SELECTED Command 8–48
8.19.13 Using the BUFFER Command 8–48
8.19.14 Reading Files into EVE 8–49
8.19.15 Writing Files from EVE 8–49
8.19.16 Example 8–49
8.19.17 Using Windows 8–49
8.19.18 Keys Used with EVE Windows 8–50
8.19.19 EVE Window Commands 8–50
8.19.20 Viewing Two Sections of One Buffer 8–51
8.19.21 Editing Two Buffers 8–51
8.20 Creating a Subprocess 8–52
8.20.1 Overview 8–52
8.20.2 Spawning 8–52
8.20.3 Example 8–52
8.20.4 Spawning to EVE from DCL 8–52
8.20.5 Example 8–52

9 Editing Text Files: Using EDT

9.1 Overview 9–1
9.1.1 References 9–1
9.2 EDT Overview 9–1
9.2.1 Editing Modes 9–1
9.2.2 Nokeypad Editing 9–2
9.3 Beginning EDT Editing Sessions 9–2
9.3.1 Invoking EDT 9–2

9.3.2 Example 9–2
9.3.3 Editing Existing Files 9–2
9.3.4 Creating Files 9–2
9.3.5 Changing Editing Modes 9–2
9.4 Entering EDT Line Commands 9–3
9.4.1 Line Editing 9–3
9.4.2 Example: Line Editing 9–3
9.4.3 Using Line Numbers 9–3
9.4.4 Example: Displaying Line Numbers 9–3
9.4.5 Specifying a Range of Lines 9–3
9.4.6 EDT Line-Mode Command Ranges 9–4
9.4.7 EDT Command-Line Symbols for Specifying Ranges 9–4
9.4.8 Canceling EDT Commands 9–5
9.5 Entering EDT Keypad Commands 9–5
9.5.1 Overview 9–5
9.5.2 Keypad Editing 9–5
9.5.3 Using the Keypad 9–5
9.5.4 Example: Using the WORD Function 9–6
9.5.5 Example: Using the CHNGCASE Function 9–6
9.6 Using Online Help in EDT 9–6
9.6.1 Getting Keypad Help 9–6
9.6.2 Getting Line Mode Help 9–6
9.6.3 Getting Nokeypad Help 9–7
9.7 Ending EDT Editing Sessions 9–7
9.7.1 Overview 9–7
9.7.2 Saving Edits 9–7
9.7.3 Examples 9–7
9.7.4 Ending EDT Sessions Without Saving Edits 9–7
9.8 Changing Editing Modes 9–7
9.8.1 Overview 9–7
9.8.2 Changing from Keypad to Line Editing 9–8
9.8.3 Changing from Line to Keypad Editing 9–8
9.8.4 Entering Line-Editing Commands from Keypad Mode 9–8
9.8.5 Example 9–8
9.9 Recovering from Interruptions 9–9
9.9.1 Restoring the Display 9–9
9.9.2 Recovering from Ctrl/Y 9–9
9.9.3 Journal Files 9–9
9.10 Summary of EDT Commands 9–9
9.10.1 Changing Editing Modes 9–9
9.10.2 Moving the Cursor 9–10
9.10.3 Inserting Text 9–11
9.10.4 Deleting and Restoring Text 9–12
9.10.5 Locating Text 9–13
9.10.6 Substituting Text 9–14
9.10.7 Moving Text 9–15
9.10.8 Indenting Text 9–15
9.10.9 Changing the Case of Text 9–16
9.10.10 Using Multiple Buffers 9–17
9.10.11 Defining Keys 9–17
9.10.12 Controlling Screen and Terminal Settings 9–18
9.10.13 Processing EDT Commands 9–19
9.10.14 Other EDT Commands 9–20

10 DIGITAL Standard Runoff (DSR): Formatting Text Files

10.1 Overview 10–1
10.1.1 References 10–1
10.2 About DSR 10–1
10.2.1 Overview 10–1
10.2.2 Formatting a File Using DSR 10–2
10.3 Entering DSR Commands 10–2
10.3.1 Creating Source Files 10–2
10.3.2 Example 10–2
10.4 Invoking DSR 10–2
10.4.1 Using the RUNOFF Command 10–2
10.4.2 Examples 10–3
10.4.3 Overriding DSR Commands or Flags 10–3
10.4.4 Example 10–3
10.4.5 RUNOFF Command Qualifiers 10–3
10.4.6 Using DSR Defaults 10–4
10.4.7 Disabling Default Settings 10–5
10.5 Creating Tables of Contents 10–5
10.5.1 How to Create a Table of Contents 10–5
10.5.2 Table of Contents Default Settings 10–5
10.5.3 Example: Creating a Table of Contents 10–6
10.5.4 DSR Qualifiers for Tailoring a Table of Contents 10–6
10.5.5 Example 10–7
10.6 Creating Indexes 10–8
10.6.1 Index Entry Formats 10–8
10.6.2 Example 10–8
10.6.3 Producing an Index 10–8
10.6.4 Default Index Settings 10–8
10.6.5 Example: Producing an Index 10–9
10.6.6 DSR Qualifiers for Tailoring an Index 10–9
10.6.7 Example: Tailoring an Index 10–10
10.7 Summary of DSR Commands 10–11
10.7.1 Page Size and Running Heads 10–11
10.7.2 Paging and Page-Number Control 10–12
10.7.3 Subpaging 10–12
10.7.4 Margin Settings 10–13
10.7.5 Filling and Justifying Text 10–13
10.7.6 Vertical Spacing 10–14
10.7.7 Horizontal Spacing 10–14
10.7.8 Paragraph Formatting 10–15
10.7.9 Text Emphasis 10–15
10.7.10 Figures 10–16
10.7.11 Lists 10–16
10.7.12 Notes and Footnotes 10–17
10.7.13 Chapters and Appendixes 10–17
10.7.14 Sections 10–18
10.7.15 Indexes 10–19
10.7.16 Tables of Contents 10–19
10.7.17 Flag Recognition Commands 10–20
10.7.18 Other DSR Commands 10–21

11 Sort/Merge Utility: Sorting and Merging Files

11.1 Overview . . . 11–1
11.1.1 Using High-Performance Sort/Merge . . . 11–1
11.1.2 About This Chapter . . . 11–1
11.1.3 References . . . 11–2
11.2 Sorting Files . . . 11–2
11.2.1 SORT Command . . . 11–2
11.2.2 Sort Keys . . . 11–2
11.2.3 Defining a Key . . . 11–3
11.2.4 Multiple Key Fields . . . 11–6
11.2.5 Identical Key Fields . . . 11–7
11.2.6 Noncharacter Data . . . 11–8
11.2.7 Output File Organization . . . 11–8
11.2.8 Sorting Process . . . 11–9
11.3 Specifying a Collating Sequence . . . 11–10
11.3.1 Examples . . . 11–13
11.4 Running Sort as a Batch Job . . . 11–13
11.4.1 Definition: Batch Jobs . . . 11–13
11.4.2 Command Procedures . . . 11–13
11.4.3 Including Input Records . . . 11–14
11.5 Merging Files . . . 11–14
11.5.1 MERGE Command . . . 11–14
11.5.2 Sorted Files . . . 11–15
11.5.3 Identical Key Fields . . . 11–16
11.6 Entering Records from a Terminal . . . 11–16
11.6.1 Procedure . . . 11–16
11.7 Using a Sort/Merge Specification File . . . 11–17
11.7.1 Overview . . . 11–17
11.7.2 Format . . . 11–17
11.7.3 Overriding a Qualifier . . . 11–18
11.7.4 Order of Qualifiers . . . 11–18
11.7.5 Including Comments . . . 11–18
11.7.6 Examples: Specification File . . . 11–18
11.8 Optimizing a Sort or Merge Operation . . . 11–23
11.8.1 Overview . . . 11–23
11.8.2 Sorting Process . . . 11–24
11.8.3 Omitting Records and Fields . . . 11–25
11.8.4 Work Files . . . 11–25
11.8.5 Working Set Extent . . . 11–26
11.9 Summary of Sort/Merge Qualifiers . . . 11–26
11.9.1 Command Qualifiers . . . 11–26
11.9.2 Input File Qualifier . . . 11–31
11.9.3 Output File Qualifiers . . . 11–32
11.9.4 Specification File Qualifiers . . . 11–35

12 Devices: Using Private Tapes and Disks

12.1 Overview . . . 12–1
12.1.1 References . . . 12–1
12.2 Devices Overview . . . 12–1
12.2.1 Device Managment . . . 12–1
12.2.2 Accessing Devices . . . 12–1
12.2.3 Device Security . . . 12–2
12.2.4 Volumes . . . 12–2

12.2.5 Setting Up Private Volumes 12–2
12.2.6 Printing Files from Private Devices 12–2
12.3 Displaying Device Information 12–2
12.3.1 Using the SHOW DEVICES Command 12–2
12.3.2 Example 12–3
12.4 Allocating Devices 12–3
12.4.1 Overview 12–3
12.4.2 Format 12–3
12.4.3 Methods of Allocating Devices 12–3
12.5 Initializing Volumes 12–4
12.5.1 Using the INITIALIZE Command 12–4
12.5.2 INITIALIZE Command Format 12–5
12.5.3 Initializing Disk Volumes 12–5
12.5.4 Initializing Magnetic Tape Volumes 12–5
12.6 Mounting Volumes 12–6
12.6.1 Overview 12–6
12.6.2 Using the MOUNT Command 12–6
12.6.3 Volume Sets 12–6
12.6.4 MOUNT Command Format 12–6
12.6.5 Requesting Operator Assistance 12–6
12.6.6 Example: Requesting Operator Assistance 12–7
12.6.7 Example: Mounting Disk Volumes 12–7
12.6.8 Mounting a Foreign Disk Volume 12–7
12.6.9 Mounting Magnetic Tape Volumes 12–7
12.6.10 Example 12–8
12.7 Accessing Files on Private Devices 12–8
12.7.1 Overview 12–8
12.7.2 Using Physical Device Names 12–8
12.7.3 Accessing Files on Volume Sets 12–8
12.7.4 Using Device Names with Commands 12–9
12.7.5 Using Logical Device Names 12–9
12.7.6 Example 12–9
12.7.7 Using Generic Device Names 12–10
12.7.8 Using VMScluster Device Names 12–10
12.7.9 VMScluster Device Name Format 12–10
12.8 Dismounting Volumes 12–11
12.8.1 Overview 12–11
12.8.2 Logical Dismounting 12–11
12.8.3 Dismounted Volumes 12–11
12.8.4 Allocated Devices 12–11

13 Logical Names: Defining Names for Devices and Files

13.1 Overview 13–1
13.1.1 References 13–1
13.2 About Logical Names 13–1
13.2.1 Usage 13–1
13.2.2 Example 13–2
13.2.3 Using Logical Names for File Input/Output (I/O) 13–2
13.2.4 Example 13–2
13.3 Using System-Defined Logical Names 13–2
13.3.1 Systemwide Logical Names 13–2
13.3.2 Examples 13–2
13.4 Creating Logical Names 13–3

13.4.1 Overview . . . 13–3
13.4.2 Using the DEFINE Command . . . 13–3
13.4.3 Examples . . . 13–3
13.4.4 Rules for Creating Logical Names . . . 13–4
13.4.5 Example . . . 13–4
13.4.6 Creating Logical Node Names . . . 13–4
13.4.7 Rules for Creating Logical Node Names . . . 13–5
13.4.8 Example . . . 13–5
13.4.9 Using Logical Node Names in File Specifications . . . 13–5
13.4.10 Example . . . 13–5
13.4.11 Overriding Access Control Strings . . . 13–5
13.4.12 Examples . . . 13–6
13.4.13 Translation Attributes . . . 13–6
13.4.14 Creating Concealed Logical Names . . . 13–6
13.4.15 Creating Terminal Logical Names . . . 13–6
13.4.16 Example: Translation Attributes . . . 13–7
13.4.17 Equating Several Logical Names to One Equivalence String . . . 13–7
13.4.18 Search Lists . . . 13–7
13.4.19 Examples . . . 13–7
13.4.20 Using Search Lists with Wildcards . . . 13–8
13.4.21 Example . . . 13–8
13.4.22 Imprecise File Specifications . . . 13–8
13.4.23 Logical Name Tables . . . 13–9
13.4.24 Using Different Logical Name Tables . . . 13–9
13.4.25 Access Modes . . . 13–9
13.4.26 Example . . . 13–10
13.4.27 User Mode . . . 13–10
13.4.28 Example . . . 13–10
13.4.29 Executive Mode . . . 13–10
13.5 Creating Logical Name Tables . . . 13–10
13.5.1 Overview . . . 13–10
13.5.2 Process Table . . . 13–10
13.5.3 Job Table . . . 13–11
13.5.4 Group Table . . . 13–11
13.5.5 System Table . . . 13–11
13.5.6 Creating Logical Names in Tables . . . 13–11
13.5.7 Process Private Logical Name Tables . . . 13–11
13.5.8 Example . . . 13–12
13.5.9 Shareable Logical Name Tables . . . 13–12
13.5.10 Adding Logical Names to Logical Name Tables . . . 13–12
13.5.11 Example . . . 13–12
13.5.12 Adding Logical Names to the Logical Name Directory . . . 13–13
13.5.13 Examples . . . 13–13
13.5.14 Defining the Protection for a Logical Name Table . . . 13–13
13.5.15 Quotas for Logical Name Tables . . . 13–14
13.5.16 Specifying Quotas . . . 13–14
13.5.17 Example . . . 13–14
13.5.18 Setting Job Table Quotas . . . 13–14
13.5.19 Using Logical Name Directory Tables . . . 13–14
13.6 Process Directory Table LNM$PROCESS_DIRECTORY . . . 13–15
13.6.1 Overview . . . 13–15
13.6.2 LNM$GROUP . . . 13–15
13.6.3 LNM$JOB . . . 13–15
13.6.4 LNM$PROCESS . . . 13–15

13.6.5 LNM$PROCESS_DIRECTORY 13–15
13.6.6 LNM$PROCESS_TABLE (Process Table) 13–16
13.6.7 LNM$JOB_xxx (Job Table) 13–16
13.7 System Directory Table LNM$SYSTEM_DIRECTORY 13–17
13.7.1 LNM$DCL_LOGICAL 13–17
13.7.2 LNM$DIRECTORIES 13–17
13.7.3 LNM$FILE_DEV 13–17
13.7.4 LNM$GROUP_xxx (Group Table) 13–18
13.7.5 LNM$JOB_xxx 13–18
13.7.6 LNM$PERMANENT_MAILBOX 13–18
13.7.7 LNM$SYSTEM 13–18
13.7.8 LNM$SYSTEM_TABLE (System Table) 13–18
13.7.9 LNM$TEMPORARY_MAILBOX 13–20
13.8 Displaying Logical Names 13–20
13.8.1 Using the SHOW LOGICAL Command 13–20
13.8.2 Examples 13–21
13.8.3 Displaying Process Permanent Files 13–21
13.8.4 Displaying the Access Mode for a Logical Name 13–21
13.8.5 Displaying Logical Name Tables 13–21
13.8.6 Example 13–21
13.8.7 Displaying Directory Table Structure 13–22
13.9 Deleting Logical Names and Logical Name Tables 13–22
13.9.1 Using the DEASSIGN Command 13–22
13.9.2 Examples 13–23
13.10 Redefining Process-Permanent Logical Names 13–23
13.10.1 Overview 13–23
13.10.2 Available Process Permanent Logical Names 13–23
13.10.3 Using the System Interactively 13–23
13.10.4 Executing Command Procedures Interactively 13–23
13.10.5 Submitting Batch Jobs 13–24
13.10.6 Nested Command Procedures 13–24
13.10.7 Opening Files 13–24
13.11 Using Process-Permanent Logical Names as File Specifications 13–24
13.11.1 Overview 13–24
13.11.2 Redefining SYS$INPUT 13–24
13.11.3 Examples 13–24
13.11.4 Redefining SYS$OUTPUT 13–25
13.11.5 Examples 13–25
13.11.6 Redefining SYS$ERROR 13–26
13.11.7 Redefining SYS$COMMAND 13–26
13.12 Logical Name Translation 13–26
13.12.1 Translation of Logical Names 13–26
13.12.2 Examples 13–27
13.12.3 Modifying Logical Name Translation 13–27
13.12.4 Example 13–27
13.12.5 Iterative Translation 13–27
13.12.6 Example 13–28
13.12.7 System Defaults During Logical Name Translation 13–28
13.12.8 Example 13–28
13.13 Search Lists 13–29
13.13.1 Search List Translation 13–29
13.13.2 Examples 13–29
13.13.3 Search Lists with Wildcards 13–30
13.13.4 Example 13–30

13.13.5 Using the RUN Command with Search Lists 13–30
13.13.6 Search Order for Multiple Search Lists 13–30
13.13.7 Examples 13–31
13.13.8 Logical Names in Multiple Tables 13–31

14 Symbols: Defining Commands and Expressions

14.1 Overview 14–1
14.1.1 References 14–1
14.2 About Symbols 14–2
14.2.1 Symbol Usage 14–2
14.2.2 Example 14–2
14.2.3 Comparing Logical Names and Symbols 14–2
14.3 Using Symbols 14–3
14.3.1 Types of Symbols 14–3
14.3.2 Defining Symbols 14–3
14.3.3 Using the Assignment Statement 14–3
14.3.4 Examples: Creating Local Symbols 14–3
14.3.5 Examples: Creating Global Symbols 14–4
14.3.6 Using Symbols to Represent DCL Commands 14–4
14.3.7 Symbol Abbreviation 14–4
14.3.8 Example 14–4
14.3.9 Defining Foreign Commands 14–5
14.3.10 Example 14–5
14.3.11 Symbol Substitution 14–5
14.3.12 Deleting Symbols 14–6
14.4 Displaying Symbols 14–6
14.4.1 Using the SHOW SYMBOL Command 14–6
14.4.2 Examples 14–6
14.5 Using Symbols with Other Symbols 14–6
14.5.1 Defining a Symbol as a Symbol 14–6
14.5.2 Example 14–6
14.5.3 Symbol Concatenation 14–7
14.5.4 Examples 14–7
14.5.5 Including Symbols in String Assignments 14–7
14.5.6 Examples 14–7
14.6 Using Symbols to Store and Manipulate Data 14–8
14.6.1 Variables 14–8
14.6.2 Expressions 14–8
14.6.3 Example 14–8
14.7 Character Strings 14–8
14.7.1 Overview 14–8
14.7.2 Types of Characters 14–8
14.7.3 Defining Character Strings 14–9
14.7.4 Examples 14–9
14.7.5 Character String Expressions 14–9
14.7.6 Character String Expression Operands 14–10
14.7.7 Examples 14–10
14.7.8 Character String Operations 14–10
14.7.9 Examples 14–10
14.7.10 Comparing Character Strings 14–11
14.7.11 Types of String Comparisons 14–11
14.7.12 Examples 14–12
14.7.13 Replacing Substrings 14–13

14.7.14 Rules for Replacing Substrings 14–13
14.7.15 Examples 14–13
14.8 Using Numeric Values and Expressions 14–14
14.8.1 Overview 14–14
14.8.2 Specifying Numbers 14–14
14.8.3 Examples 14–14
14.8.4 Internal Storage of Numbers 14–15
14.8.5 Numeric Expressions 14–15
14.8.6 Integer Operands 14–15
14.8.7 Performing Arithmetic Operations 14–16
14.8.8 Examples 14–16
14.8.9 Comparing Numbers 14–17
14.8.10 Examples 14–17
14.8.11 Performing Numeric Overlays 14–18
14.8.12 Rules for Using Numeric Overlays 14–18
14.8.13 Example 14–18
14.9 Using Logical Values and Expressions 14–18
14.9.1 Logical Operations 14–18
14.9.2 Example 14–19
14.9.3 Logical Expressions 14–19
14.9.4 Uses of Logical Expressions 14–19
14.9.5 Examples 14–20
14.9.6 Logical Operation Results 14–20
14.9.7 Using Values Returned by Lexical Functions 14–21
14.9.8 Lexical Function Format 14–21
14.9.9 Rules for Using Lexical Functions 14–21
14.9.10 Evaluation of Lexical Functions 14–21
14.9.11 Example 14–22
14.9.12 Lexical Function Usage 14–22
14.9.13 Examples 14–22
14.9.14 Using the F$DIRECTORY Lexical Function 14–23
14.9.15 Order of Operations 14–23
14.9.16 Overriding Default Precedence 14–24
14.9.17 Example 14–24
14.9.18 Evaluating Data Types 14–24
14.9.19 Example 14–24
14.9.20 Values of Expressions 14–24
14.10 Converting Value Types in Expressions 14–25
14.10.1 Overview 14–25
14.10.2 Converting Expressions 14–25
14.10.3 Converting Strings to Integers 14–25
14.10.4 Examples 14–26
14.10.5 Converting Integers to Strings 14–26
14.11 Understanding Symbol Tables 14–26
14.11.1 Overview 14–26
14.11.2 Local Symbol Tables 14–26
14.11.3 Parameters in Local Symbol Tables 14–26
14.11.4 Global Symbol Tables 14–27
14.11.5 $STATUS Reserved Global Symbol 14–27
14.11.6 $SEVERITY Reserved Global Symbol 14–27
14.11.7 $RESTART Reserved Global Symbol 14–27
14.11.8 Symbol Table Search Order 14–27
14.12 Masking the Value of Symbols 14–28
14.12.1 SET SYMBOL Command 14–28

14.12.2 Verb String Translation 14–28
14.12.3 Symbol Scoping State 14–28
14.12.4 Local Symbol Scope 14–28
14.12.5 Global Symbol Scope 14–29
14.13 Understanding Symbol Substitution 14–29
14.13.1 Overview 14–29
14.13.2 Automatic Evaluation of Symbols 14–29
14.13.3 Examples 14–29
14.13.4 Forced Symbol Substitution 14–30
14.13.5 Precedence of Symbol Substitution 14–30
14.13.6 Example 14–31
14.13.7 Symbol Substitution Operators 14–31
14.13.8 The Apostrophe (') 14–31
14.13.9 Examples 14–32
14.13.10 The Ampersand (&) 14–32
14.13.11 Examples 14–32
14.13.12 Rules for Using Ampersands 14–33
14.14 The Three Phases of Command Processing 14–33
14.14.1 Overview 14–33
14.14.2 Phase 1: Command Input Scanning 14–34
14.14.3 Phase 2: Command Parsing 14–34
14.14.4 Phase 3: Expression Evaluation 14–34
14.14.5 Symbol Substitution on Data Lines 14–34
14.14.6 Example 14–34
14.14.7 Repetitive and Iterative Substitution 14–34
14.14.8 Phase 1 Substitutions 14–35
14.14.9 Example 14–35
14.14.10 Phase 2 Substitution 14–35
14.14.11 Example 14–35
14.14.12 Phase 3 Substitution 14–36
14.14.13 Examples 14–36
14.14.14 Undefined Symbols 14–37
14.14.15 Examples 14–37
14.15 An Alternative to Using Symbols: Automatic Foreign Commands 14–38
14.15.1 Overview 14–38
14.15.2 How Automatic Foreign Commands Work 14–38
14.15.3 Example 14–38
14.15.4 Example 14–38
14.15.5 Using Automatic Foreign Commands 14–39
14.15.6 Caution: Defining DCL$PATH 14–40
14.15.7 Restrictions 14–40

15 Introduction to Command Procedures: Programming with DCL

15.1 Overview 15–1
15.1.1 Resources 15–1
15.1.2 Types of Command Procedures 15–1

Basic Information for Writing Command Procedures 15–3
15.2 Overview 15–3
15.2.1 Creating Command Procedures 15–3
15.2.2 File Type 15–3
15.2.3 Writing Commands 15–3
15.2.4 Writing Command Lines 15–3
15.2.5 Use of Dollar Signs ($) 15–4

15.3 Use of Labels 15–4
15.3.1 Rules for Using Labels 15–4
15.3.2 Labels in Local Symbol Tables 15–4
15.3.3 Duplicate Labels 15–5
15.4 Using Comments in Command Procedures 15–5
15.4.1 Suggested Uses of Comments 15–5
15.4.2 Rules for Using Comments 15–5

How to Write Command Procedures 15–7
15.5 Overview 15–7
15.5.1 Before You Begin 15–7
15.5.2 Definitions 15–7
15.5.3 About This Section 15–7
15.5.4 The Steps for Writing Command Procedures 15–7
15.6 Step 1: Design the Command Procedure 15–8
15.6.1 How to Design Command Procedures 15–8
15.6.2 Deciding Which Tasks to Perform 15–8
15.6.3 Variables 15–8
15.6.4 Conditionals 15–8
15.6.5 Design Decisions 15–8
15.6.6 Ordering Commands 15–9
15.7 Step 2: Assign Variables and Test Conditionals 15–9
15.7.1 Overview 15–9
15.7.2 How to Assign Variables and Test Conditionals 15–9
15.7.3 Using the INQUIRE Command 15–9
15.7.4 Preserving Literal Characters 15–10
15.7.5 Example: Using the INQUIRE Command 15–10
15.7.6 Testing Conditionals 15–10
15.7.7 IF... THEN Commands 15–10
15.7.8 Writing Program Stubs 15–10
15.7.9 Example: Assigning Variables and Testing Conditionals 15–11
15.8 Step 3: Add Loops 15–12
15.8.1 Overview 15–12
15.8.2 Writing a Loop 15–12
15.8.3 Example: GET_COM_LOOP 15–12
15.9 Step 4: End the Command Procedure 15–13
15.9.1 How to End Command Procedures 15–13
15.9.2 Using the EXIT Command 15–13
15.9.3 Examples 15–13
15.9.4 When to Use the EXIT Command 15–14
15.9.5 Using the STOP Command 15–14
15.9.6 Example 15–14
15.10 Step 5: Test and Debug the Program Logic 15–14
15.10.1 What Needs to Be Tested? 15–14
15.10.2 How to Test and Debug Command Procedures 15–14
15.10.3 Example: Testing the Program Logic 15–14
15.10.4 Debugging Command Procedures 15–15
15.10.5 Example: Debugging Using the SET VERIFY Command 15–15
15.10.6 Example: Debugging Using the SET PREFIX Command 15–16
15.10.7 Example: Debugging Using the SHOW SYMBOL Command 15–16
15.10.8 Enabling Verification During Execution 15–16
15.11 Step 6: Add Clean-Up Tasks 15–16
15.11.1 Cleaning Up 15–16
15.11.2 How to Add Clean Up Tasks 15–17
15.11.3 Closing Files 15–17

15.11.4 Example 15–17
15.11.5 Deleting Temporary or Extraneous Files 15–17
15.11.6 Saving and Restoring Defaults 15–18
15.11.7 Example 15–18
15.11.8 Commonly Changed Process Characteristics 15–18
15.11.9 Ensuring Cleanup Operations Are Performed 15–18
15.12 Step 7: Complete the Command Procedure 15–19
15.12.1 How to Complete Command Procedures 15–19
15.12.2 Example: Replacing a Program Stub with Commands 15–19
15.13 Example: CLEANUP.COM Command Procedure 15–19

Executing Command Procedures 15–21
15.14 Methods of Executing Command Procedures 15–21
15.14.1 Overview 15–21
15.15 Executing Command Procedures from Within Other Command Procedures 15–21
15.15.1 Command Line 15–21
15.15.2 Example 15–21
15.16 Executing Command Procedures on Remote Nodes 15–21
15.16.1 Remote Nodes 15–21
15.16.2 Command Line 15–22
15.16.3 Example: SHOWUSERS Command Procedure 15–22
15.16.4 Example: Executing SHOWUSERS 15–22
15.16.5 Security Note 15–22
15.17 Executing Command Procedures as Qualifiers or Parameters to DCL Commands 15–22
15.17.1 Overview 15–22
15.17.2 Command Line 15–22
15.17.3 Examples 15–23
15.17.4 Restrictions 15–23
15.18 Executing Command Procedures Interactively 15–23
15.18.1 Command Line 15–23
15.18.2 Example 15–23
15.18.3 Defining Symbols 15–23
15.18.4 Example 15–24
15.18.5 Redirecting Interactive Output 15–24
15.18.6 Example 15–24
15.18.7 /OUTPUT Qualifier Restriction 15–24
15.19 Executing Command Procedures as Batch Jobs 15–24
15.19.1 Overview 15–24
15.19.2 Submitting Batch Jobs 15–24
15.19.3 Example 15–25
15.19.4 Remote Batch Jobs 15–25
15.19.5 Restarting Batch Jobs 15–25
15.19.6 Using $RESTART and BATCH$RESTART 15–25
15.19.7 Example 15–26
15.20 Executing Command Procedures on Disk and Tape Volumes 15–27
15.20.1 Executing on Private Disk Volumes 15–27
15.20.2 Executing on Tape Volumes 15–27

Exiting, Interrupting and Error Handling Command Procedures 15–29
15.21 Exiting from Command Procedures 15–29
15.21.1 Definition: Command Level 15–29
15.21.2 Methods of Exiting 15–29
15.21.3 Exiting with the EXIT Command 15–29

15.21.4 Example 15–29
15.21.5 Exiting with the STOP Command 15–29
15.21.6 Exiting with Ctrl/Y 15–30
15.21.7 Example 15–30
15.21.8 Exit-Handling Routines 15–30
15.22 Handling Error Conditions 15–30
15.22.1 Handling Errors 15–30
15.22.2 Handling Label Errors 15–30
15.22.3 Example: Displaying Errors 15–30
15.22.4 Default Error Actions 15–31
15.23 Other Methods of Error Handling 15–31
15.23.1 ON Command 15–31
15.23.2 ON Command Format 15–32
15.23.3 Example: Using the ON Command 15–32
15.23.4 Example: Resuming After an Error 15–32
15.23.5 Figure: ON Command Actions 15–33
15.23.6 For Additional Information 15–33
15.24 Using the SET NOON Command 15–34
15.24.1 Disabling Error Checking 15–34
15.24.2 Example: Disabling Error Checking 15–34
15.24.3 Example: Checking the Value of $STATUS 15–34
15.24.4 Command Levels 15–34
15.25 Handling Ctrl/Y Interruptions 15–35
15.25.1 Ctrl/Y Interruptions 15–35
15.25.2 Stopping Command Procedures 15–35
15.25.3 After Ctrl/Y Is Entered 15–35
15.25.4 Note: Ctrl/Y on Exiting 15–36
15.25.5 Stopping Privileged Images 15–36
15.26 Setting Ctrl/Y Action Routines 15–36
15.26.1 Using the ON Command 15–36
15.26.2 When Ctrl/Y Is Pressed 15–37
15.26.3 Effects of Entering Ctrl/Y 15–37
15.26.4 Example: Using the ON Command 15–37
15.26.5 Figure: Flow of Execution Following Ctrl/Y Action 15–38
15.26.6 Figure: Ctrl/Y in Nested Procedures 15–39
15.27 Disabling and Enabling Ctrl/Y Interruptions 15–40
15.27.1 Using SET NOCONTROL=Y 15–40
15.27.2 Using the SET CONTROL=Y Command 15–40
15.27.3 Disabling Ctrl/Y Interrupts 15–40
15.27.4 Example: Using SET [NO]CONTROL=Y Commands 15–41
15.28 Detecting Errors in Command Procedures Using Condition Codes 15–41
15.28.1 Overview 15–41
15.28.2 Displaying Condition Codes ($STATUS) 15–41
15.28.3 Example 15–42
15.28.4 Condition Codes with the EXIT Command 15–42
15.28.5 Example 15–42
15.28.6 Severity Levels 15–43
15.28.7 Testing for Successful Completion 15–43
15.29 Using Commands That Do Not Set $STATUS 15–44

Login Command Procedures 15–45
15.30 Login Command Procedures 15–45
15.30.1 Overview 15–45
15.30.2 Systemwide Login Command Procedures 15–45
15.30.3 Creating Systemwide Login Command Procedures 15–45

15.30.4 Personal Login Command Procedures 15–45
15.30.5 Login Command Procedures in Captive Accounts 15–46
15.30.6 For More Information 15–46

16 Complex Command Procedures: Programming with DCL

16.1 Overview 16–1
16.1.1 Who Should Read This Chapter 16–2
16.1.2 For More Information 16–2
16.2 Performing Command Procedure Input 16–2
16.2.1 Overview 16–2
16.2.2 Including Input Data 16–2
16.2.3 Example 16–2
16.2.4 Restrictions to Including Data in Command Procedures 16–2
16.2.5 Other Methods of Inputting Data 16–3
16.3 Using Parameters to Pass Data 16–3
16.3.1 Rules for Passing Parameters 16–3
16.3.2 How Parameters Are Passed 16–3
16.3.3 Specifying Parameters as Integers 16–3
16.3.4 Specifying Parameters as Character Strings 16–3
16.3.5 Specifying Parameters as Symbols 16–4
16.3.6 Examples 16–4
16.3.7 Specifying Parameters as Null Values 16–4
16.4 Using Parameters to Pass Data to Batch Jobs 16–5
16.4.1 The SUBMIT Command 16–5
16.4.2 Example 16–5
16.4.3 Example: Executing Multiple Command Procedures 16–5
16.4.4 Defining SYS$INPUT 16–5
16.5 Using Parameters to Pass Data to Nested Command Procedures 16–5
16.5.1 Parameters in Nested Procedures 16–5
16.5.2 Example 16–5
16.6 Using the READ Command to Prompt for Data 16–6
16.6.1 Prompting for User Input 16–6
16.6.2 Comparing the INQUIRE and READ Commands 16–6
16.6.3 The READ Command 16–6
16.6.4 Examples 16–7
16.6.5 Note: Symbols and Batch Jobs 16–7
16.7 Using the SYS$INPUT Logical Name to Obtain Data 16–7
16.7.1 Overview 16–7
16.7.2 Redefining SYS$INPUT as Your Terminal 16–7
16.7.3 Examples 16–7
16.7.4 Defining SYS$INPUT as a Separate File 16–8
16.7.5 Including Programs in Data Files 16–8
16.7.6 Example 16–8
16.8 Performing Command Procedure Output 16–9
16.8.1 Overview 16–9
16.9 Writing Data to Terminals 16–9
16.9.1 Using the TYPE Command 16–9
16.9.2 Example 16–9
16.9.3 Using the WRITE Command 16–9
16.9.4 Character Strings as Literal Text 16–9
16.9.5 Quotation Marks in Character Strings 16–10
16.9.6 Character String Concatenation 16–10
16.9.7 Forcing Symbol Substitutions in Character Strings 16–10

16.10 Redirecting Output from Commands and Images 16–10
16.10.1 Redirecting SYS$OUTPUT 16–10
16.10.2 Example: Defining SYS$OUTPUT as a File 16–10
16.10.3 Example: Defining SYS$OUTPUT as a Null Device 16–11
16.10.4 Using the /USER_MODE Qualifier 16–11
16.10.5 Commands Performed Within the Command Interpreter 16–11
16.10.6 Example: Using the DEASSIGN Command 16–12
16.11 Returning Data from Command Procedures 16–12
16.11.1 Global Symbols and Logical Names 16–12
16.11.2 Example: Passing Values with Global Symbols 16–12
16.11.3 Example: Passing Values with Logical Names 16–13
16.11.4 Example: Logical Names with Calling Procedures 16–13
16.12 Redirecting Error Messages 16–13
16.12.1 Redefining SYS$ERROR 16–13
16.12.2 Error Messages 16–14
16.12.3 Example: Redefining SYS$ERROR 16–14
16.12.4 Suppressing System Error Messages 16–14
16.12.5 Example 16–14
16.13 Reading and Writing Files (File I/O) 16–15
16.13.1 Reading and Writing Files 16–15
16.13.2 Note: Process Permanent Files 16–15
16.13.3 In This Section 16–15
16.14 Using the OPEN command 16–16
16.14.1 Overview 16–16
16.14.2 Opening Files 16–16
16.14.3 Example 16–16
16.14.4 Note: Logical Names 16–16
16.14.5 Specifying Files 16–16
16.14.6 Reading Files 16–16
16.14.7 Example 16–17
16.14.8 Writing Files 16–17
16.14.9 Example 16–17
16.14.10 Appending Records 16–17
16.14.11 Example 16–17
16.14.12 Reading and Writing Files 16–18
16.14.13 Example 16–18
16.15 Writing to Files 16–18
16.15.1 How to Write to Files 16–18
16.15.2 Example 16–19
16.15.3 Creating Files with Unique File Names 16–19
16.15.4 Example 16–19
16.16 Using the WRITE Command 16–20
16.16.1 Specifying Data 16–20
16.16.2 Example 16–20
16.16.3 Using the /SYMBOL Qualifier 16–21
16.16.4 Using the /UPDATE Qualifier 16–21
16.17 Using the Read Command 16–21
16.17.1 Reading from a File 16–21
16.17.2 Example 16–22
16.17.3 Specifying Symbols 16–22
16.17.4 Using the /END_OF_FILE Qualifier 16–23
16.17.5 Using the READ Command in Loops 16–23
16.17.6 Using the /INDEX and /KEY Qualifiers 16–23
16.17.7 Using the /DELETE Qualifier 16–23

16.17.8 For Additional Information 16–23
16.18 Using the CLOSE Command 16–23
16.18.1 Closing Files 16–23
16.18.2 Example 16–24
16.19 Modifying Files 16–24
16.19.1 In This Section 16–24
16.19.2 Updating records 16–24
16.19.3 Example 16–25
16.20 Creating Output Files 16–25
16.20.1 Making Extensive Changes to Files 16–25
16.20.2 Note: Opening Files 16–26
16.20.3 Creating Output Files 16–26
16.20.4 Modifying Records 16–26
16.20.5 Examples: Modifying Records 16–27
16.20.6 Example: Creating Output Files 16–27
16.21 Appending Records to Files 16–28
16.21.1 Appending Records to Files 16–28
16.21.2 Example 16–28
16.22 Handling File I/O Errors 16–28
16.22.1 Using the /ERROR Qualifier 16–28
16.22.2 Example 16–29
16.22.3 Displaying F$STATUS 16–29
16.22.4 Example 16–29
16.22.5 Default Error Actions 16–29

Techniques for Controlling Execution Flow 16–31
16.22.6 Overview 16–31
16.22.7 In This Section 16–31
16.23 Using the IF Command 16–31
16.23.1 IF Command 16–31
16.23.2 Formats for the IF Command 16–32
16.23.3 Using the THEN Command 16–32
16.23.4 Example 16–32
16.23.5 Using the ELSE Command 16–32
16.23.6 Example 16–32
16.24 Using Command Blocks 16–33
16.24.1 Executing Command Blocks 16–33
16.24.2 Command Blocks and THEN Commands 16–33
16.24.3 Example: Command Blocks and THEN Commands 16–33
16.24.4 Example: Command Blocks And Else Commands 16–34
16.24.5 IF Command Restrictions 16–34
16.24.6 True Expressions 16–34
16.24.7 False Expressions 16–35
16.24.8 Writing Expressions 16–35
16.24.9 Example: Using Logical Operators 16–35
16.24.10 Using Symbols 16–35
16.24.11 Example: Comparing Integers and Strings 16–36
16.24.12 Example: Executing Other Command Procedures 16–36
16.24.13 GOTO Command 16–36
16.24.14 Example: Using the GOTO Command 16–37
16.24.15 Example: Incorrectly Placed GOTO Command 16–37
16.24.16 Avoiding Reexecution 16–37
16.24.17 The $RESTART Global Symbol 16–37
16.24.18 Example 16–38
16.24.19 Symbols in System Failures 16–38

16.25 Using the GOSUB Command 16–38
16.25.1 GOSUB Command 16–38
16.25.2 RETURN Command 16–39
16.25.3 Example: Using the GOSUB Command 16–39
16.26 Using the CALL Command 16–40
16.26.1 Command Levels 16–40
16.26.2 CALL Command 16–40
16.26.3 CALL Command Rules 16–40
16.26.4 CALL Command Defaults 16–40
16.26.5 Beginning and Ending Subroutines 16–41
16.26.6 Scope Defining Restrictions 16–41
16.26.7 Examples: Scope Defining Restrictions 16–41
16.26.8 Example: Calling a Subroutine 16–42
16.27 Writing Case Statements 16–43
16.27.1 Case Statements 16–43
16.27.2 Listing Labels 16–43
16.27.3 Example 16–43
16.27.4 Writing the Case Statement 16–43
16.27.5 Example 16–43
16.27.6 Writing Command Blocks 16–44
16.27.7 Example 16–44
16.27.8 Writing Loops 16–44
16.27.9 Example 16–45
16.27.10 Using Counters 16–45
16.27.11 Example 16–46
16.27.12 Using F$ELEMENT 16–46
16.27.13 Examples 16–46

17 Lexical Functions: Obtaining and Manipulating Information

17.1 Overview 17–1
17.1.1 References 17–1
17.2 About Lexical Functions 17–1
17.2.1 Overview 17–1
17.2.2 Information Manipulation 17–2
17.3 Obtaining Information About Your Process 17–2
17.3.1 Process Lexical Functions 17–2
17.3.2 Commonly Changed Process Characteristics 17–3
17.3.3 Saving Process Characteristics 17–4
17.3.4 Changing Verification Settings 17–4
17.3.5 Examples 17–4
17.3.6 Time Stamping 17–5
17.3.7 Changing Default File Protection 17–5
17.4 Obtaining Information About the System 17–6
17.4.1 System Information 17–6
17.4.2 Determining Your VMScluster Node Name 17–6
17.4.3 Example 17–7
17.4.4 Obtaining Queue Information 17–7
17.4.5 Example 17–7
17.4.6 Obtaining Process Information 17–7
17.4.7 Examples 17–7
17.4.8 F$CONTEXT Lexical Function 17–8
17.4.9 Example 17–8
17.5 Obtaining Information About Files and Devices 17–10

17.5.1 Overview 17–10
17.5.2 Searching for Devices 17–10
17.5.3 Example 17–10
17.5.4 Searching for a File in a Directory 17–10
17.5.5 Deleting Old Versions of Files 17–11
17.6 Translating Logical Names 17–11
17.6.1 Overview 17–11
17.6.2 Obsolete Function 17–11
17.6.3 Variables in Command Procedures 17–12
17.6.4 Examples 17–12
17.7 Manipulating Strings 17–12
17.7.1 Overview 17–12
17.7.2 Determining Presence of Strings or Characters 17–13
17.7.3 Example 17–13
17.7.4 Extracting Parts of Strings 17–13
17.7.5 Example 17–14
17.7.6 Formatting Output Strings 17–15
17.7.7 Examples 17–15
17.8 Manipulating Data Types 17–17
17.8.1 Overview 17–17
17.8.2 Converting Data Types 17–17
17.8.3 Evaluating Expressions 17–17
17.8.4 Example 17–18
17.8.5 Determining Whether a Symbol Exists 17–18

18 Processes and Batch Jobs: Using the OpenVMS Environment

18.1 Overview 18–1
18.1.1 Resources 18–1
18.1.2 How Processes Are Created 18–1
18.2 Interpreting Your Process Context 18–1
18.2.1 Displaying Process Context 18–1
18.2.2 Example 18–2
18.2.3 Using Detached Processes 18–4
18.3 Using Subprocesses 18–4
18.3.1 Overview 18–4
18.3.2 Job Trees 18–4
18.3.3 Using Subprocesses to Spawn Tasks 18–4
18.3.4 Using Subprocesses to Perform Multiple Tasks 18–5
18.3.5 Creating a Subprocess 18–5
18.3.6 Example 18–5
18.3.7 Exiting from a Subprocess 18–6
18.3.8 Example 18–6
18.3.9 Subprocess Context 18–7
18.3.10 Parent Process Characteristics Not Inherited by the Subprocess 18–7
18.3.11 Preventing Inheritance 18–7
18.3.12 Transfer of Control 18–8
18.4 Connecting to Disconnected Processes on Virtual Terminals 18–8
18.4.1 Overview 18–8
18.4.2 Terminal Disconnections 18–8
18.4.3 Process Reconnections 18–8
18.4.4 Removing Disconnected Processes 18–9
18.4.5 Managing Disconnected Processes 18–9
18.5 Working with Batch Jobs 18–10

18.5.1 Overview . . . 18–10
18.5.2 Submitting Batch Jobs . . . 18–10
18.5.3 Using the SUBMIT Command . . . 18–10
18.5.4 Example . . . 18–10
18.5.5 Job Entries . . . 18–10
18.5.6 Specifying Start Times . . . 18–11
18.5.7 Ensuring Correct Access of Files . . . 18–11
18.5.8 Checking for Batch Jobs in Your Login Command Procedure . . . 18–11
18.5.9 Example . . . 18–12
18.5.10 Submitting Multiple Command Procedures . . . 18–12
18.5.11 Examples . . . 18–12
18.5.12 Passing Data to Batch Jobs . . . 18–13
18.5.13 Examples . . . 18–13
18.5.14 Passing Confidential Information . . . 18–13
18.5.15 Control of Batch Job Output . . . 18–14
18.5.16 Log File Output . . . 18–14
18.5.17 Modifying Log File Names . . . 18–14
18.5.18 Preventing Printing . . . 18–14
18.5.19 Time Stamping . . . 18–14
18.5.20 Saving Log Files . . . 18–15
18.5.21 Example . . . 18–15
18.5.22 Reading the Log File . . . 18–15
18.5.23 Including Command Output in the Batch Job Log . . . 18–15
18.5.24 Examples . . . 18–16
18.5.25 Changing Batch Job Characteristics . . . 18–16
18.5.26 Example . . . 18–16
18.5.27 Changes You Can Make to Batch Entries . . . 18–16
18.5.28 SUBMIT Command Qualifiers . . . 18–17
18.5.29 Displaying Jobs in Batch Queues . . . 18–18
18.5.30 Examples . . . 18–18
18.5.31 Deleting and Stopping Batch Jobs . . . 18–19
18.5.32 Deleting a Batch Job . . . 18–19
18.5.33 Restarting Batch Jobs . . . 18–19
18.5.34 Example . . . 18–20
18.5.35 Synchronizing Batch Job Execution . . . 18–20
18.5.36 Examples . . . 18–20
18.5.37 Using the WAIT Command . . . 18–21
18.5.38 Example: Using the WAIT Command . . . 18–21

19 Security: Controlling Access to Resources

19.1 Overview . . . 19–1
19.1.1 Resources . . . 19–1
19.1.2 Security Features . . . 19–1
19.2 Displaying the Rights Identifiers of Your Process . . . 19–2
19.2.1 Overview . . . 19–2
19.2.2 Example: Displaying Rights Identifiers . . . 19–2
19.3 Security Profile of Objects . . . 19–3
19.3.1 Overview . . . 19–3
19.3.2 Displaying a Security Profile . . . 19–3
19.3.3 Modifying a Security Profile . . . 19–3
19.4 Interpreting Protection Codes . . . 19–4
19.4.1 Protection Code Format . . . 19–4
19.4.2 Production Categories . . . 19–4

19.4.3 Access List . . . 19–5
19.5 Default File Protection . . . 19–5
19.5.1 Overview . . . 19–5
19.5.2 Default UIC protection . . . 19–5
19.5.3 Default ACL protection . . . 19–6
19.5.4 Example . . . 19–6
19.5.5 Renaming Files . . . 19–6
19.5.6 Explicit File Protection . . . 19–6
19.5.7 Examples . . . 19–6
19.6 Accessing Files Across Networks . . . 19–7
19.6.1 Access Control Strings . . . 19–7
19.6.2 Access Control String Format . . . 19–7
19.6.3 Caution: ACLs and Security . . . 19–7
19.6.4 Protecting Access Control Strings . . . 19–7
19.6.5 Using Proxy Login Accounts to Protect Passwords . . . 19–7
19.6.6 Initiating Proxy Logins . . . 19–8
19.6.7 Examples . . . 19–8
19.6.8 General Access Proxy Accounts . . . 19–9
19.6.9 Specify Proxy Accounts . . . 19–9
19.7 Auditing Access to Your Account and Files . . . 19–10
19.7.1 Overview . . . 19–10
19.7.2 Observing Your Last Login Time . . . 19–10
19.7.3 Asking Your Security Administrator to Enable Auditing . . . 19–10
19.7.4 Events That Can Trigger Security Alarms . . . 19–11
19.7.5 Example: Unauthorized Access to an Audited File . . . 19–11
19.7.6 Security Audit Log Files . . . 19–11
19.7.7 Adding ACEs to Sensitive Files . . . 19–12
19.7.8 Example . . . 19–12

A Customizing EVE

A.1 Overview . . . A–1
A.1.1 References . . . A–1
A.1.2 About Startup Files . . . A–1
A.1.3 Tailoring the Standard Editor . . . A–2
A.2 Defining EVE Keys . . . A–2
A.2.1 Overview . . . A–2
A.2.2 Undefinable Keys . . . A–2
A.2.3 Obsolete Keys . . . A–3
A.2.4 Defining Keys to Execute EVE Commands . . . A–3
A.2.5 Using the DEFINE KEY Command . . . A–3
A.2.6 Key Name Abbreviations . . . A–4
A.2.7 Defining Control Keys . . . A–5
A.2.8 Differences Between EVE and DECTPU . . . A–5
A.2.9 EVE Key Names . . . A–5
A.2.10 Undefinable Keys . . . A–6
A.2.11 Keys That You Should Not Define . . . A–6
A.2.12 Defining the GOLD Key . . . A–7
A.2.13 Using the GOLD Key . . . A–7
A.2.14 GOLD Key Combinations . . . A–7
A.2.15 Tutorial: Creating GOLD Key Definitions . . . A–8
A.2.16 Removing GOLD Keys . . . A–9
A.2.17 Defining GOLD Keys in Initialization Files . . . A–9
A.3 Learn Sequences . . . A–9

A.3.1 Overview A–9
A.3.2 Defining Learn Sequences A–9
A.3.3 Tutorial: Defining Learn Sequences A–10
A.4 Setting and Saving Attributes A–10
A.4.1 Overview A–10
A.4.2 EVE Default Settings A–10
A.4.3 EVE Default Buffer-Specific Settings A–12
A.4.4 Default Direction A–13
A.4.5 Saving Attributes A–13
A.4.6 Initialization Files A–14
A.4.7 Example A–14
A.4.8 Saving Attributes A–14
A.4.9 EVE Commands for Saving Attributes A–15
A.4.10 Saving Attributes While Editing A–17
A.4.11 Disabling Prompting A–17
A.4.12 Saving Attributes in a Section File A–17
A.4.13 Creating Section Files A–17
A.4.14 Example: Creating Section Files A–18
A.4.15 EVE Settings for Saving Attributes A–18
A.4.16 Specifying Section Files A–18
A.4.17 Saving Attributes in Command Files A–19
A.4.18 Example: EVE Generated Code A–19
A.4.19 Example: Saving Attributes in Command Files A–19
A.4.20 Default Command Files A–20
A.4.21 Setting a Default Command File A–20
A.4.22 Saving EVE Default Attributes A–20
A.5 Using DECTPU Within EVE A–20
A.5.1 Overview A–20
A.5.2 Creating DECTPU Command Files A–21
A.5.3 Using a DECTPU Debugging Package A–21
A.5.4 Specifying a Debug File A–21
A.6 Using DECTPU Procedures to Extend EVE A–22
A.6.1 Overview A–22
A.6.2 Entering DECTPU Commands A–22
A.6.3 Writing DECTPU Procedures A–22
A.6.4 Rules for Writing EVE Command Procedures A–22
A.6.5 Example: Defining a Global Variable A–23
A.6.6 Compiling DECTPU Procedures A–24
A.6.7 Example: Creating DECTPU Procedures A–24
A.7 Creating Section Files A–25
A.7.1 Overview A–25
A.7.2 Startup Section Files A–25
A.7.3 Modifying Section Files A–25
A.7.4 Using Section Files A–26
A.7.5 Default Section Files A–26
A.7.6 Section Files Execution A–26
A.8 Creating Command Files A–26
A.8.1 Overview A–26
A.8.2 Command File Usage A–26
A.8.3 Setting Editing Defaults A–27
A.8.4 Converting Command Files A–27
A.8.5 Adding Functions to the EVE Editor A–27
A.8.6 Default Files Specification A–27
A.8.7 Rules for Writing Command Files A–28

A.8.8 Example: EVE Command File A–29
A.9 Creating Initialization Files A–32
A.9.1 Overview A–32
A.9.2 Rules for Creating Initialization Files A–32
A.9.3 Example A–33
A.9.4 Specifying Initialization Files A–33
A.9.5 Default Initialization File A–33
A.9.6 Commands That Define the Environment A–33
A.10 Saving Your Customizations in Startup Files A–34
A.10.1 Overview A–34
A.10.2 Types of Startup Files A–34
A.10.3 EVE Commands for Saving Attributes A–34
A.11 Saving Attributes A–36
A.11.1 Overview A–36
A.11.2 Example: Saving Attributes A–36
A.11.3 Disabling Prompting A–36
A.11.4 Using Customizations A–36
A.12 Key Definitions in EVE Startup Files A–37
A.12.1 Key Definitions A–37
A.12.2 Global Settings-1 . . . . . A–37
A.12.3 Global Settings-2 . . . . . A–38
A.12.4 Buffer Settings A–38
A.12.5 DECTPU Procedures A–38
A.12.6 Saving in a Section File A–38
A.12.7 Example A–39
A.12.8 EVE Settings for Saving Attributes A–39
A.12.9 Specifying Section Files A–39
A.12.10 Saving in a Command File A–40
A.12.11 Setting a Default Command File A–40
A.12.12 Specifying Startup Command Files A–41
A.13 Converting from EDT to EVE A–41
A.13.1 Overview A–41
A.13.2 Using the SET KEYPAD EDT Command A–41
A.13.3 Defining Keys for EVE Commands A–41
A.13.4 Example: Defining Keys for EVE Commands A–42
A.13.5 Setting Bound Cursor Motion A–42
A.13.6 Setting the Right Margin for Wrapping Text A–42
A.13.7 Setting Scroll Margins for Moving the Cursor A–42
A.13.8 Setting Searches for Exact Case A–43
A.13.9 Converting EDT Macros to DECTPU Procedures A–43
A.13.10 Examples: Converting EDT Macros to DECTPU Procedures A–43
A.13.11 Converting EDT Nokeypad Statements to DECTPU Procedures A–44
A.13.12 Using the WPS Keypad Ruler Key A–44
A.13.13 Tutorial: Using the WPS Keypad Ruler A–45

B Character Sets

B.1 Overview B–1
B.2 DEC Multinational Character Set B–1
B.2.1 Overview B–1
B.2.2 ASCII Character Set B–1
B.2.3 Example B–1
B.2.4 ASCII Character Set, Part 1 . . . . . B–2
B.2.5 ASCII Character Set, Part 2 . . . . . B–3

B.3 DCL Character Set B–4

C Annotated Command Procedures

C.1 Overview C–1
C.2 CONVERT.COM Command Procedure C–1
C.2.1 Introduction C–1
C.2.2 Example: CONVERT.COM C–1
C.2.3 Notes for CONVERT.COM Command Procedure C–3
C.2.4 Sample Execution for CONVERT.COM Command Procedure C–5
C.3 REMINDER.COM Command Procedure C–5
C.3.1 Overview C–5
C.3.2 Example: REMINDER.COM C–5
C.3.3 Notes for REMINDER.COM Command Procedure C–7
C.3.4 Sample Execution for REMINDER.COM Command Procedure C–8
C.4 DIR.COM Command Procedure C–9
C.4.1 Overview C–9
C.4.2 Example: DIR.COM C–9
C.4.3 Notes for DIR.COM Command Procedure C–10
C.4.4 Sample Execution for DIR.COM Command Procedure C–11
C.5 SYS.COM Command Procedure C–11
C.5.1 Overview C–11
C.5.2 Example: SYS.COM C–11
C.5.3 Notes for SYS.COM Command Procedure C–12
C.5.4 Sample Execution for SYS.COM Command Procedure C–13
C.6 GETPARMS.COM Command Procedure C–13
C.6.1 Overview C–13
C.6.2 Example: GETPARMS.COM C–13
C.6.3 Notes for GETPARMS.COM Command Procedure C–14
C.6.4 Sample Execution for GETPARMS.COM Command Procedure C–15
C.7 EDITALL.COM Command Procedure C–15
C.7.1 Overview C–15
C.7.2 Example: EDITALL.COM C–15
C.7.3 Notes for EDITALL.COM Command Procedure C–16
C.7.4 Sample Execution for EDITALL.COM Command Procedure C–17
C.8 MAILEDIT.COM Command Procedure C–17
C.8.1 Overview C–17
C.8.2 Example: MAILEDIT.COM C–18
C.8.3 Notes for MAILEDIT.COM Command Procedure C–18
C.8.4 Example: Execution for MAILEDIT.COM Command Procedure C–18
C.9 FORTUSER.COM Command Procedure C–19
C.9.1 Overview C–19
C.9.2 Example: FORTUSER.COM C–19
C.9.3 Notes for FORTUSER.COM Command Procedure C–21
C.9.4 Sample Execution for FORTUSER.COM Command Procedure C–22
C.10 LISTER.COM Command Procedure C–23
C.10.1 Overview C–23
C.10.2 Example: LISTER.COM C–23
C.10.3 Notes for LISTER.COM Command Procedure C–24
C.10.4 Sample Execution for LISTER.COM Command Procedure C–25
C.11 CALC.COM Command Procedure C–25
C.11.1 Overview C–25
C.11.2 Example: CALC.COM C–25
C.11.3 Notes for CALC.COM Command Procedure C–26

C.11.4 Sample Execution for CALC.COM Command Procedure C–27
C.12 BATCH.COM Command Procedure C–27
C.12.1 Overview C–27
C.12.2 Example: BATCH.COM C–27
C.12.3 Notes for BATCH.COM Command Procedure C–30
C.12.4 Sample Execution for BATCH.COM Command Procedure C–31
C.13 COMPILE_FILE.COM Command Procedure C–31
C.13.1 Overview C–31
C.13.2 Example: COMPILE_FILE.COM C–32
C.13.3 Notes for COMPILE_FILE.COM Command Procedure C–33
C.13.4 Example: Execution for COMPILE_FILE.COM Command Procedure C–34

D Terminal Keys

D.1 LK201 Keyboard D–1
D.2 VT100 Terminal Series D–3

Glossary

Index

Examples

A–1 EVE-Generated Code for Saving Attributes in a Command File A–19

Figures

5–1 Directory Structure 5–2
6–1 Organizing Mail 6–15
6–2 Mail Utility Keypad 6–21
8–1 EVE Keys—VT200, VT300, and VT400 Series Terminals 8–6
8–2 EVE Keys—VT100 Series Terminals 8–7
11–1 List Sorted in Ascending Order 11–3
11–2 Record Fields in a List 11–5
11–3 Sorting by Key Field 11–6
11–4 Sorting with Default Key Records 11–6
11–5 Sorting with Multiple Key Fields 11–7
11–6 Sorting with Multiple Key Fields (Ascending and Descending Order) 11–7
11–7 Sorting with Identical Key Fields 11–8
11–8 Output from Using a Specification File 11–19
15–1 ON Command Actions 15–33
18–1 Synchronizing Batch Job Execution 18–21

Tables

8–1	EVE Commands That Move the Cursor	8–10
8–2	EVE Editing Keys for Erasing and Restoring Text	8–16
8–3	EVE Commands for Erasing and Restoring Text	8–17
8–4	EVE Commands for Locating Text in a Buffer	8–27
8–5	Buffer-Change Journal File Names	8–39
11–1	/KEY Qualifier Values	11–3
12–1	Allocating a Device	12–3
14–1	String Comparisons	14–11
14–2	Numeric Comparisons	14–17
A–1	EVE Commands for Setting Attributes	A–14
B–1	DCL Character Set	B–4

Preface

Intended Audience

This manual is intended for all users of the OpenVMS operating system.

A **system manager** performs the administrative tasks that create and maintain an efficient computing environment. If you are a system manager or want to understand system management concepts and procedures, refer to the *OpenVMS System Manager's Manual*.

Discontinued Manuals

This document includes information from the following documents that have been discontinued:

- *VMS Mail Utility Manual*
- *VMS Sort/Merge Utility Manual*
- *Introduction to VMS*
- *Guide to Using VMS Command Procedures*
- *VMS DCL Concepts Manual*
- *Guide to VMS Text Processing*
- *Guide to VMS Files and Devices*
- *VMS Phone Utility Manual*

Associated Documents

For more information, refer to the following manuals:

- The *Overview of OpenVMS Documentation* provides an overview of the OpenVMS documentation set.
- The *OpenVMS DCL Dictionary* provides complete descriptions of all DIGITAL Command Language (DCL) commands and lexical functions.
- The *Guide to the Extensible Versatile Editor* provides detailed descriptions of how to use the Extensible Versatile Editor (EVE).
- The *Extensible Versatile Editor Reference Manual* provides a summary of EVE commands.
- The *DEC Text Processing Utility Reference Manual* provides information on the DECTPU programming language.
- The *OpenVMS EDT Reference Manual* describes the EDT editor and the EDT commands for each editing mode.

- The *OpenVMS DIGITAL Standard Runoff Reference Manual* describes DIGITAL Standard Runoff (DSR). It includes descriptions of all the DSR commands, flags, and control characters.
- The *OpenVMS Guide to System Security* provides detailed information on how to create a secure system.
- The *OpenVMS System Manager's Manual* provides information on managing your system.
- The *OpenVMS National Character Set Utility Manual* provides information about the National character set (NCS) utility.
- The *OpenVMS System Management Utilities Reference Manual* describes the utilities and commands that are used by system managers.
- The *OpenVMS Command Definition, Librarian, and Message Utilities Manual* provides information on how to define foreign commands using the Command Definition utility.

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

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

Document Structure

Introduction

This manual contains 19 chapters, 4 appendixes, and a glossary. Each chapter describes concepts and procedures for performing computing tasks. Basic information is presented first within each chapter; more complex concepts and procedures are presented last.

Getting Started

Refer to the following chapters to help you get started using the OpenVMS operating system:

- Chapter 1

  Introduction: OpenVMS Concepts and Definitions describes basic concepts about the OpenVMS operating system and its components. The chapter serves as an introduction to the rest of the manual.

- Chapter 2

  Getting Started: Interacting with the OpenVMS Operating System describes how to log in and log out of the system, how to change your password, and how to get help.

- Chapter 3

  The DIGITAL Command Language: Interacting with the System describes how to use the DIGITAL Command Language (DCL). It includes a summary of keypad key combinations that let you enter and edit DCL commands.

- Chapter 4

  Files: Storing Information describes files and how you can use them to store information. It includes examples for creating, copying, renaming, displaying, deleting, protecting, and printing files.

- Chapter 5

  Directories: Organizing and Managing Files describes how to use directories to organize and manage files.

Communicating with Other Users

Refer to the following chapters to learn about communicating with other users:

- Chapter 6

  Mail: Communicating with Other Users describes how to use the Mail utility (MAIL) to communicate with other users on your system or on any other computer that is connected to your system with the DECnet for OpenVMS network. The chapter includes a sample mail message; step-by-step instructions for reading, sending, replying to, forwarding, and organizing mail messages; and a summary of Mail commands.

- Chapter 7

  Phone: Communicating with Other Users describes how to use the Phone utility (PHONE) to communicate with other users on your system or on any other computer that is connected to your system with the DECnet for OpenVMS network.)

Manipulating Text and Records

Refer to the following chapters to learn about text processing and record sorting:

- Chapter 8

  Editing Text Files: Using EVE describes EVE, an interactive text editor that is included with the OpenVMS operating system. The chapter describes how to use EVE to create and edit new files or to edit existing files. It includes summaries of EVE commands.

- Chapter 9

  Editing Text Files: Using EDT describes EDT, an interactive text editor that preceded EVE and is still included with the OpenVMS operating system. The chapter describes how to begin and end an EDT session, how to enter EDT commands, and how to get help in EDT. It includes a summary of EDT commands.

- Chapter 10

  DIGITAL Standard Runoff (DSR): Formatting Text Files describes the DIGITAL Standard Runoff (DSR) text-formatting facility. The chapter includes a summary of DSR commands.

- Chapter 11

  Sort/Merge Utility: Sorting and Merging Files describes how to use the Sort/Merge utility (SORT/MERGE) to sort records from one or more input files or to merge files that have been sorted. The chapter includes a summary of Sort/Merge command qualifiers.

Using Devices

Refer to the following chapter to learn about devices:

- Chapter 12

  Devices: Using Private Tapes and Disks describes how to reserve tapes or disks for private use. Unlike devices that are shared by a group of users, private devices might not be set up and maintained by a system manager.

Logical Names and Symbols

Refer to the following chapters to learn about logical names and symbols:

- Chapter 13

  Logical Names: Defining Names for Devices and Files describes how to create and use logical names to represent files, directories, and devices. The chapter also summarizes the logical names created by the system.

- Chapter 14

  Symbols: Defining Commands and Expressions describes how to use symbols to represent commands, character strings, and numeric data. The chapter also describes how to combine symbols into expressions to manipulate the values that the symbols represent.

Programming

Refer to the following chapters to learn about writing programs and using programming functions:

- Chapter 15

  Introduction to Command Procedures: Programming with DCL describes basic techinques used in writing command procedures, which are files that contain DCL commands and data lines used by DCL commands.

- Chapter 16

  Complex Command Procedures: Programming with DCL describes advanced techniques used in writing command procedures.

- Chapter 17

  Lexical Functions: Obtaining and Manipulating Information describes how to use lexical functions within a command procedure to get information about your process or the system environment.

Managing Processes

Refer to the following chapter to learn about managing processes:

- Chapter 18

  Processes and Batch Jobs: Using the OpenVMS Environment describes processes, which are environments created by the OpenVMS operating system that let you interact with the system. The chapter describes how and when to use subprocesses, programs, and batch jobs.

Ensuring Security

Refer to the following chapter to learn about security:

- Chapter 19

  Security: Controlling Access to Resources describes general security issues such as controlling access to protected objects and accessing data on remote systems.

Reference Sections

The following information is provided for reference:

- Appendix A

  Customizing EVE contains information on customizing your EVE editing environment.

- Appendix B

  Character Sets describes the DEC Multinational character set and the DCL character set.

- Appendix C

  Annotated Command Procedures contains complete command procedures that demonstrate the concepts and techniques discussed in Chapters 15, 16, and 17.

- Appendix D

  Terminal Keys describes how the OpenVMS operating system responds when various keys and control characters are pressed on an LK201 keyboard (VT200 series and later terminals, and workstations) or on a VT100 series terminal.

- Glossary

  The *Glossary* provides a list of terms used in this manual.

Reader's Comments

Digital welcomes your comments on this manual.

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

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

How To Order Additional Documentation

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

Telephone and Direct Mail Orders

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

ZK–7654A–GE

Terminology

References to the Operating System

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

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

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

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

Cluster References

In this manual, discussions that refer to VMScluster environments apply to both VAXcluster systems that include only VAX nodes and VMScluster systems that include at least one Alpha node, unless indicated otherwise.

Examples

The display examples for some commands described in this manual may differ from the actual output on your system. However, when the behavior of a command differs significantly between OpenVMS VAX and OpenVMS Alpha, that behavior is described in text and rendered, as appropriate, in separate examples.

Conventions

This section describes the conventions used in this manual.

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x* or GOLD *x*	A sequence such as PF1 *x* or GOLD *x* indicates that you must first press and release the key labeled PF1 or GOLD and then press and release another key or a pointing device button. GOLD key sequences can also have a slash (/), dash (–), or underscore (_) as a delimiter in EVE commands.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	Horizontal ellipsis points in examples indicate one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	Vertical ellipsis points indicate the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification or in the syntax of a substring specification in an assignment statement.)

{ }	In command format descriptions, braces indicate a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`struct`	Monospace type in text identifies the following C programming language elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

1

Introduction: OpenVMS Concepts and Definitions

1.1 Overview

This chapter describes basic concepts about the OpenVMS operating system and its components. It includes introductory information on the following topics:

- Logging in to the system
- Networks
- DIGITAL Command Language (DCL)
- Files and directories
- OpenVMS utilities
- Devices
- Logical names
- Symbols
- Command procedures
- Lexical functions
- Processes and programs
- System security

1.1.1 Overview of OpenVMS

OpenVMS is an interactive virtual memory **operating system**. While you are logged in to the computer, you and the system conduct a dialogue using the **DIGITAL Command Language (DCL)**. You use DCL by entering **commands**, which the system reads and translates. You enter a command by typing it from your **keyboard** and pressing the Return key; the system responds by executing the command or by displaying an error message on the screen if it cannot interpret what you entered.

1.1.2 Differences in Your Local Environment

Note that this manual covers standard DCL commands only. System managers at your site may tailor your system to support the local environment. A system manager might decide to:

- Use a different command language interpreter
- Change the default action of some standard DCL commands so that they do not reflect what is described in the OpenVMS documentation

- Disable some DCL commands
- Alter some system defaults, such as the DCL prompt

1.1.3 References

Additional information about the commands discussed in this chapter can be obtained from:

- The *OpenVMS DCL Dictionary*

  Contains complete information on DCL commands, lexical functions, qualifiers, and syntax.

- Online Help

  For information about obtaining online Help, refer to section Section 2.11 in this manual or enter the HELP command at the DCL prompt.

1.2 Logging In to the System

1.2.1 Accounts

To interact with the operating system, you must log in to a user **account**. An account is a name or number that identifies you to the system when you log in. That name or number tells the system where your files are stored and the type of access you have to other files.

Your system manager (or whoever authorizes system use at your installation) usually sets up accounts and grants privileges according to your needs. The type of access rights and privileges enabled for your account determine whether you have access to files, images, or utilities that might affect system performance or other users.

1.2.2 Access Requirements

To access your account, you need to enter your user name and **password**. Your system manager usually provides you with your user name and initial password. Your user name identifies you to the system and distinguishes you from other users. In many cases, a user name is your first or last name. Your password is for your protection. If you maintain its secrecy, other users cannot use system resources under your user name.

1.2.3 Logging In

Logging in consists of gaining access to the system and identifying yourself as an authorized user. When you log in, the system creates an environment from which you can enter commands. This environment is called your **process**.

Chapter 2 describes how to log in to and out of the system.

1.3 Networks

1.3.1 Overview

When computer systems are linked together, they form a **network**. Operating systems in a DECnet for OpenVMS network are able to communicate with each other and share information and resources. Each system in a network is called a network **node** and is identified by a unique node name.

1.3.2 Network Nodes

When you are logged in to a network node, you can communicate with other nodes in the network. The node at which you are logged in is called the **local node**; other nodes on the network are called **remote nodes**. If you have access to an account on a remote node, you can log in to that account from your local node and perform tasks on that node while remaining connected to your local node.

Chapter 2 describes how to log in to a remote node. Additional tasks you can perform on remote nodes are described in the appropriate chapters of this manual.

1.3.3 Executing Programs Over Networks

Because of support provided by DECnet software, **programs** can execute across the network as if they were executing locally. Because DECnet software is integrated within the operating system, it is easy to write programs that access remote files. To access a remote file in an application program, you need only include the name of the remote node and any required access control information in the file specification.

1.3.4 Task-to-Task Communication

Task-to-task communications, a feature common to all DECnet implementations, allows two application programs running on the same or different operating systems to communicate with each other regardless of the programming languages used. Examples of network applications are distributed processing applications, transaction processing applications, and applications providing connection to servers.

1.3.5 Proxy Acounts

In the examples of remote operations in this manual, proxy accounts enable users to perform operations on remote systems. Proxy accounts are one way users can access remote systems. For additional ways to access remote systems, see the *OpenVMS System Manager's Manual*.

1.4 DIGITAL Command Language (DCL)

1.4.1 Overview

DCL (DIGITAL Command Language) is a set of English-like instructions that tell the operating system to perform specific operations. DCL provides you with over 200 commands and functions to use in communicating with the operating system to accomplish computing tasks.

1.4.2 Usage Modes

You can use DCL in the following two modes:

- Interactive

  In **interactive mode**, you enter commands from your **terminal**. One command has to finish executing before you can enter another.

- Batch

  In batch mode, the system creates another process to execute commands on your behalf. A **batch job** is a command procedure or program that is submitted to the operating system for execution as a separate user process. After you submit the command procedure for batch execution, you can continue to use your terminal interactively.

  Batch jobs and network processes use DCL in batch mode. See Chapter 18 for more information about processes.

1.4.3 Types of DCL Commands

When you enter a DCL command, it is read and translated by the DCL interpreter. The way the command interpreter responds to a command is determined by the type of command entered. The three types of DCL commands are as follows:

- Built-in commands

  These commands are built in to the DCL interpreter and are executed internally.

- Commands that invoke programs

  DCL calls another program to execute the command rather than executing it internally. The program invoked to execute a command is referred to as a **command image**. This command image can be either an interactive program, a **utility** (such as Mail), or a noninteractive program (such as COPY).

- Foreign commands

  A symbol that executes an image is referred to as a **foreign command**. A foreign command executes an image whose name is not recognized by the command interpreter as a DCL command. Refer to Chapter 14 for complete information on symbols.

1.4.4 DCL Command Line

DCL, like any language, has its own vocabulary and usage rules. The vocabulary consists of commands, **parameters**, and **qualifiers**, which are put together in a way that DCL can interpret. The way in which the parts of a command line are put together is referred to as the command line **syntax**.

1.4.5 DCL Command Line Format

A DCL command line contains the following format:

```
[$] command [[/qualifier[=value]]...]  [[parameter[=value][/qualifier...]]...]
```

Note

Items in brackets [ ] are optional and might not be required by a specific command.

For a complete description of the components of a DCL command line, see Section 3.3.2.

1.4.6 Lexical Functions

Lexical functions are command language constructs that the DCL interpreter evaluates and substitutes before it interprets a **command string**. Chapter 17 discusses lexical functions in more detail.

1.5 Files and Directories

1.5.1 Definition: File

A **file** is a system object that contains information. This information can be machine-readable **data** that the computer understands. It can also be text you enter and manipulate. The text in the file might be the text of a document, a program, or a list of addresses. You can examine the data in these files by displaying the files on a terminal screen or by printing them on paper.

Chapter 4 describes how to create and organize files to store information.

1.5.2 Definition: Directory

A **directory** is a special kind of file that contains the names and locations of files; files are listed in directories. For example, when the system manager creates a user account for you (see Section 1.2), you automatically have a directory with the same name as your user name. If your user name is JONES, the directory is [JONES].

Chapter 5 describes how to use directories to organize and manage files.

1.5.3 Hardware

Directory files are stored on **disks**. Disks are one type of **hardware** device that the operating system uses to store information.

1.5.4 File Specifications

Every file must have a **file name** or **file type** to identify it to both the system and you. A file also has a **version number**. You can have several versions of a file. Unless you specify a version number, the system uses the highest existing version number of a file. When you edit a file, the system saves the original file and produces a modified **output file**. By default, the output file has the same name and type as the original but the version number is incremented by one.

The file name, file type, and version number form a **file specification**. A full file specification:

- Completely describes the access path the system uses to locate and identify a file
- Can include the directory in which the file is located and the network node on which the file resides
- Is also known as a network file specification

1.5.5 Directory Structures

Each disk contains a main directory, which can be set up by a system manager or by the system itself. This main directory is called the master file directory (MFD) and contains a list of user file directories (UFDs). **UFDs** are files in the master file directory that point to top-level directories. Your top-level directory is usually your **login** or **default directory**. Unless your account has been modified to do otherwise, the system automatically places you in your top-level directory when you log in.

In most cases, a UFD exists for each user on the system. It contains the names of and pointers to files cataloged in a user's directory. Chapter 5 contains more information about directory structures.

1.5.6 Subdirectories

A **subdirectory** is a directory file within another directory or subdirectory file. You can have up to seven levels of subdirectories. Subdirectories let you organize files into meaningful groups. For example, you might have one subdirectory that contains memos and another subdirectory for status reports.

Like a directory, a subdirectory contains names and pointers for the files cataloged within it. It can contain an entry for another subdirectory, which can contain an entry for another subdirectory, and so on to seven levels of subdirectories. This structure (a top-level directory plus a maximum of seven levels of subdirectories) is called a **hierarchical directory structure**.

1.6 OpenVMS Utilities

1.6.1 The Mail Utility

The OpenVMS Mail utility (MAIL) lets you send messages to and receive messages from other users on your system or on any computer that is connected to your system by DECnet software.

Chapter 6 describes how to use Mail.

1.6.2 The OpenVMS Phone Utility

The OpenVMS Phone utility (PHONE) lets you communicate with other users on your system or on any computer that is connected to your system by DECnet software.

Chapter 7 describes how to use Phone.

1.6.3 Text Editors

Text **editors** allow you to create and modify text files. With a text editor, you can enter text from a keyboard and modify the text using text editing commands. For example, you can type in data for a report and then rearrange sections, duplicate information, substitute phrases, or format text. You can use text editors to create and modify source files for programming languages (such as DEC C for OpenVMS or VAX BASIC) or text formatters (such as VAX DOCUMENT or DIGITAL Standard Runoff). The operating system supports several text editors. Chapter 8 describes how to use EVE, and Chapter 9 describes how to use EDT.

1.6.4 DIGITAL Standard Runoff (DSR)

DIGITAL Standard Runoff (DSR) is a text formatter that processes source files into formatted text and intermediate files, and creates tables of contents and indexes. You use a text editor to create a source file, to which you should give the file type .RNO. This file contains text, DSR formatting commands, flags (special instruction characters you insert), and control characters.

Chapter 10 describes how to use DSR and lists DSR commands.

1.6.5 The Sort/Merge Utility

The OpenVMS Sort/Merge (SORT/MERGE) utility can be invoked in two ways: by using the SORT command or by using the MERGE command. When you invoke the Sort/Merge utility with the DCL command SORT, it sorts records from one or more **input files**, according to the fields you select, and generates one reordered output file. You can use the Sort/Merge utility to reorder records in a file (or files) so that they are in alphabetic or numeric order and either ascending or descending order.

When you invoke the Sort/Merge utility with the DCL command MERGE, it combines up to ten previously sorted files into one ordered output file.

For information about using the Sort/Merge utility, see Chapter 11.

1.7 Devices

1.7.1 Mass Storage Devices

Mass storage devices, such as disks and **magnetic tapes**, save the contents of files on a magnetic medium. Files saved this way can be accessed, updated, modified, or reused at any time.

1.7.2 Record Oriented Devices

Record-oriented devices, such as terminals, printers, mailboxes, and card readers **read** and **write** only single physical units of data at a time and do not provide online storage of the data. (Printers and card readers are also called unit-record devices.)

1.7.3 Disks and Magnetic Tapes

The files you commonly access are stored on disks or magnetic tapes. Your user file directory (UFD) and your default directory with all your files and subdirectories are located on a disk. You can use a file specification that contains directory information only if the file is located on a disk. Magnetic tapes do not have directory structures. To obtain a file stored on tape, use a file specification that contains only file information.

1.7.4 For Additional Information

- Chapter 5 describes how to access files that are not on your default device.
- Chapter 12 describes how to use **private volumes**, which are tape and disk devices that are not available systemwide.

1.8 Logical Names

1.8.1 Overview

A logical name is a name equated to an equivalence string or to a list of equivalence strings. When you define a logical name, you equate one character **string** to an **equivalence string**, which is usually a full or partial file specification, another logical name, or any other character string. Once you equate a logical name to one or more equivalence strings, you can use the logical name to refer to those equivalence strings. For example, you might assign a logical name to your **default disk** and directory. Logical names serve two main functions: they increase readability and file independence.

1.8.2 Readability

You can define commonly used files, directories, and devices with short, meaningful logical names. Such names are easier to remember and type than the full file specifications. You can define names that you use frequently in your login command procedure. A system manager can define names that most users on your system use frequently in the site-specific system startup command procedure.

1.8.3 File Independence

You can use logical names to keep your programs and command procedures independent of physical file specifications. For example, if a command procedure references the logical name ACCOUNTS, you can equate ACCOUNTS to any file on any disk before executing

1.8.4 For Additional Information

Chapter 13 contains information about logical name tables and describes how to use logical names.

1.9 Symbols

1.9.1 Definition: Symbols

Symbols are names that represent numeric, character, or logical values. When you use a symbol in a DCL command line, DCL uses the value you assign to the symbol. By defining a symbol as a command line, you can execute the command by typing only the symbol name.

1.9.2 When to Use Symbols

Entering DCL command lines that include parameters, multiple qualifiers, and values can make for much typing and can be time-consuming. To simplify your interaction with DCL and to save time, you can establish **symbols** to use in place of command names and entire command strings you type frequently.

You can also use symbols in command procedures to collect, store, and manipulate certain types of data.

1.9.3 For Additional Information

Chapter 14 describes how to use symbols in DCL commands and command procedures.

1.10 Command Procedures

1.10.1 Definition: Command Procedure

A **command procedure** is a file that contains a series of DCL commands. Some simple command procedures might contain only one or two DCL commands; complex command procedures can function as sophisticated computer programs. When a command procedure is executed, the DCL interpreter reads the file and executes the commands it contains.

1.10.2 System Login Command Procedures

If your system manager has set up a system **login command procedure**, it is executed when you log in. A system login command procedure lets your system manager ensure that certain commands are always executed when you and other users on the system log in.

1.10.3 Personal Login Command Procedures

After executing the system login command procedure, the system executes your personal login command procedure, if one exists. Your personal login command procedure lets you customize your computing environment. The commands contained in it are executed every time you log in. When you log in, the system automatically executes up to two login command procedures (the systemwide login command procedure and your own login command procedure, if it exists).

1.10.4 Creating Login Command Procedures

The person who sets up your account might have placed a login command procedure in your top-level directory. If a login command procedure is not in your top-level directory, you can create one yourself, name it LOGIN.COM and place it in your top-level directory. Unless your system manager tells you otherwise, the LOGIN.COM file that you create will be executed when you log in. A sample personal login command procedure is included in Chapter 15.

1.10.5 For Additional Information

Appendix C contains several complete command procedures.

1.11 Lexical Functions

1.11.1 Definition: Lexical Functions

Lexical functions return information to a command line or command procedure. The information returned can be about your process, the system, files and devices, logical names, strings, or data types. Lexical functions are identified by the prefix F$.

1.11.2 Usage

You can use lexical functions in any context in which you normally use symbols or expressions. In command procedures, you can use lexical functions to translate logical names, to perform character string manipulations, and to determine the current processing mode of the procedure.

1.11.3 For Additional Information

Chapter 17 describes how to use lexical functions to obtain and manipulate information within a command procedure.

1.12 Processes and Programs

1.12.1 User Authorization Files (UAFs)

The system obtains the characteristics that are unique to your process from the **user authorization file (UAF)**. The UAF lists those users permitted to access the system and defines the characteristics for each user's process. The system manager usually maintains the UAF. It is within your process that the system executes your programs (also called images or executable images) one at a time.

1.12.2 Processes

A process can be a **detached process** (a process that is independent of other processes) or a **subprocess** (a process that is dependent on another process for its existence and resources). Your main process, also called your parent process, is a detached process.

Chapter 18 describes how to use processes to perform computing tasks.

1.12.3 Programs

A program, also called an **image** or an **executable image**, is a file that contains instructions and data in machine-readable format. Some programs are associated with and invoked by a DCL command. For example, when you type the DCL command COPY, the system executes the program SYS$SYSTEM:COPY.EXE. Some programs are invoked by entering the DCL command RUN followed by the program name.

1.12.4 Creating Image Files

Image files can be supplied by the operating system or by you and usually have the other window file type .EXE. You cannot examine an image file with the DCL commands TYPE, PRINT, or EDIT because image files do not consist of **ASCII** characters. (Text files contain ASCII characters, which are a standard method of representing the alphabet, punctuation marks, numerals, and other special symbols.)

Chapter 18 contains more information about using programs.

1.13 System Security

1.13.1 Overview

Each system site has unique security requirements. For this reason, every site should have a system security policy that outlines physical and **software** security requirements for system managers and users. The *OpenVMS Guide to System Security* describes the security features available with the operating system and the tasks that system managers can perform to maintain account and system security.

1.13.2 Protected Objects

To ensure system security, the OpenVMS operating system controls both access to the system and access to any object that contains shareable information. These objects, such as devices, volumes, logical name tables, files, and queues, are known as **protected objects**. All protected objects list a set of access requirements that specify who has a right to access the object in a given manner.

1.13.3 For Additional Information

- Chapter 2 describes password management and account security.
- Chapter 4 and Chapter 5 describe how users can protect their files and directories from unauthorized access.
- Chapter 19 describes general security issues such as controlling access to protected objects and accessing data on remote systems.

2

Getting Started: Interacting with the OpenVMS Operating System

2.1 Overview

This chapter describes the following basic information that you need to know to interact with the OpenVMS operating system:

- Logging in to the system
- Choosing passwords for your account
- Reading informational messages
- Types of logins and login classes
- Login failures
- Changing passwords
- Password and account expiration times
- Guidelines for protecting your password
- Recognizing system responses
- Getting help about the system
- Logging out of the system
- Logging out without compromising system security

2.1.1 Site-Specific Procedures

The way you log in and log out of the OpenVMS operating system depends on how the system is set up at your site. This section provides a general description of logging in to and out of the operating system. Check with your system manager for the procedures specific to your site.

2.1.2 References

- Complete descriptions of all commands referenced in this chapter can be found in the *OpenVMS DCL Dictionary* and in online help.
- Additional information on passwords and OpenVMS security can be found in *OpenVMS Guide to System Security*.

2.2 Logging In to the System

2.2.1 Introduction

You need two pieces of information to log in to the system: a user name and a password. Your system manager usually sets up accounts and gives you a user name and initial password (see Section 2.3.3).

2.2.2 How to Log In

To log in to the system, use the following procedure:

Step	Task
1	Make sure your terminal is plugged in and the power is turned on.
2	Press the Return key to signal the system that you want to log in. You might need to press Return several times. The system displays a prompt for your user name: `Username:`
3	Enter your user name and press Return. You have approximately 30 seconds to do this; otherwise, the system ''times out.'' If a **timeout** occurs, you must start the login procedure again. The system displays your user name on the screen as you type it. For example: `Username: CASEY` Return The system prompts you for your password: `Password:`
4	Enter your password and press Return. The system does not display your password, which is sometimes referred to as ''no echo.''
5	Depending on how your system manager has set up your account, you might be required to enter a second password or to use an automatically generated password (see Section 2.3.6).

2.2.3 Successful Logins

If your login is successful, the system displays a dollar sign ($) in the left margin of your screen. The dollar sign is the default DCL **prompt**; it indicates that the system is ready to use.

2.2.4 Example

The following example shows a successful login:

```
Return
Username:  CASEY Return
Password:                Return
        Welcome to OpenVMS on node MARS
    Last interactive login on Friday, 11-DEC-1995 08:41
    Last non-interactive login on Thursday, 10-DEC-1995 11:05
$
```

2.2.5 Login Errors

If you make a mistake entering your user name or password or if your password has expired, the system displays the message `User authorization failure` and you are not logged in. If you make a mistake, press Return and try again. If your password has expired, you need to change your password; the system will automatically display the Set Password: prompt. See Section 2.7 for information on changing your password in this instance. If you have any other problems logging in, get help from the person who set up your account.

2.3 Choosing Passwords for Your Account

2.3.1 Guidelines

To choose a secure password, use the following guidelines:

- Include both numbers and letters in the password. Although a 6-character password that contains only letters is fairly secure, a 6-character password with both letters and numbers is much more secure.
- Choose passwords that contain 6 to 10 characters. Adequate length makes passwords more secure. You can choose a password as long as 32 characters.
- Do not select passwords from a dictionary or from your native language.
- Avoid choosing words readily associated with your computer site or yourself, such as the name of a product or the model of your car.
- Choose new passwords each time. Do not reuse old ones.

Your system manager or security administrator may set up additional restrictions, for example, not allowing passwords with fewer than 10 characters or not allowing repeats of passwords.

2.3.2 Secure and High-Risk Passwords

The following table provides examples of secure passwords and high-risk passwords (words that others might easily guess):

Secure Passwords	High-Risk Passwords
Nonsense syllables: aladaskgam eojfuvcue joxtyois	Words with a strong personal association: your name the name of a loved one the name of your pet the name of your town the name of your automobile
A mixed string: 492_weid $924spa zu_$rags	A work-related term: your company name a special project your work group name

2.3.3 Obtaining Your Initial Password

Typically, when you learn that an account has been created for you on the system, you are told whether a user password is required. If user passwords are in effect, your system manager will usually assign a specific password for your first login. This password has been placed in the system user authorization file (UAF) with other information about how your account can be used.

It is inadvisable to have passwords that others could easily guess. Ask the person creating the account for you to specify a password that is difficult to guess. If you have no control over the password you are given, you might be given a password that is the same as your first name. If so, change it immediately after you log in. (The use of first or last names as passwords is a practice so well known that it is undesirable from a security standpoint.)

At the time your account is created, you should also be told a minimum length for your password and whether you can choose your new password or whether the system generates the password for you.

2.3.4 Changing Your Initial Password

Log in to your account soon after it is created to change your password. If there is a time lapse from the moment your account is created until your first login, other users might log in to your account successfully, gaining a chance to damage the system. Similarly, if you neglect to change the password or are unable to do so, the system remains vulnerable. Possible damage depends largely on what other security measures are in effect.

2.3.5 Restrictions on Passwords

The system screens passwords for acceptability, as follows:

- It automatically compares new passwords to a system dictionary. This helps to ensure that a password is not a native language word.
- It maintains a history list of your old passwords and compares each new password to this list to be sure that you do not reuse a password.
- It enforces a minimum password length, which the system manager specifies in your UAF record.

The system rejects any passwords that it finds in a system dictionary, that you have used before, and that are shorter than the minimum password length specified in your UAF.

2.3.6 Types of Passwords

There are several types of passwords recognized by the OpenVMS operating system:

- **User password**

  Required for most accounts. After entering your user name, you are prompted for a password. If the account requires both primary and secondary passwords, two passwords must be entered.

- **System password**

  Controls access to particular terminals and is required at the discretion of the security administrator. System passwords are usually necessary to control access to terminals that might be targets for unauthorized use, such as dialup and public terminal lines.

- **Primary password**

  The first of two passwords to be entered for an account requiring both primary and secondary passwords.

- **Secondary password**

  The second of two passwords to be entered for an account requiring both primary and secondary passwords. The secondary password provides an additional level of security on user accounts. Typically, the primary user does not know the secondary password; a supervisor or other key person must be present to supply it. For certain applications, the supervisor may also decide to remain present while the account is in use. Thus, secondary passwords facilitate controlled logins and the actions taken after a login.

  Secondary passwords can be time-consuming and inconvenient. They are justified only at sites with maximum security requirements. An example of an account that justifies dual passwords would be one that bypasses normal access controls to permit emergency repair to a database.

2.3.7 Entering a System Password

Your security administrator will tell you if you must specify a system password to log in to one or more of the terminals designated for your use. Ask your security administrator for the current system password, how often it changes, and how to obtain the new system password when it does change.

2.3.8 How to Enter a System Password

To specify a system password, do the following:

Step	Task
1	Press the Return key until the terminal responds with the recognition character, which is commonly a bell. Return <bell>

Step	Task
2	Enter the system password and press Return: Return There is no prompt and the system does not display the characters you type. If you fail to specify the correct system password, the system does not notify you. (Initially, you might think the system is malfunctioning unless you know that a system password is required at that terminal.) If you do not receive a response from the system, assume that you have entered the wrong password and try again.

3	When you enter the correct system password, you receive the system announcement message, if there is one, followed by the Username: prompt. For example:
	
	`MAPLE - A member of the Forest Cluster`
	`Unauthorized Access is Prohibited`
	
	`Username:`

2.3.9 Entering a Secondary Password

Your security administrator decides whether to require the use of secondary passwords for your account at the time your account is created. When your account requires primary and secondary passwords, you need two passwords to log in. Minimum password length, which the security administrator specifies in your UAF, applies to both passwords.

As with a single password login, the system allots a limited amount of time for the entire login. If you do not enter a secondary password in time, the login period expires.

2.3.10 Example

The following example shows a login that requires primary and secondary passwords:

```
     WILLOW - A member of the Forest Cluster
         Welcome to OpenVMS on node WILLOW

Username: RWOODS
Password:            Return
Password:            Return
    Last interactive login on Friday, 11-DEC-1995 10:22
$
```

2.3.11 Password Requirements for Different Types of Accounts

Four types of user accounts are available on OpenVMS systems:

- Accounts secured with passwords that you or the security administrator change periodically. This account type is the most common.

- Accounts that always require passwords but prohibit you from changing the password. By locking the password (setting the LOCKPWD flag in the UAF), the security administrator controls all changes made to the password.
- **Restricted accounts** limit your use of the system and sometimes require a password.
- **Open accounts** require no password. When you log in to an open account, the system does not prompt you for a password and you do not need to enter one. You can begin entering commands immediately. Because open accounts allow anyone to gain access to the system, they are used only at sites with minimal security requirements.

2.4 Reading Informational Messages

2.4.1 Introduction

When you log in from a terminal that is directly connected to a computer, the OpenVMS system displays informational system messages.

2.4.2 Example: Local Login Messages

```
WILLOW - A member of the Forest Cluster                          ❶
        Unlawful Access is Prohibited

Username:  RWOODS
Password:
    You have the following disconnected process:                 ❷
Terminal   Process name    Image name
VTA52:     RWOODS          (none)
Connect to above listed process [YES]: NO
         Welcome to OpenVMS on node WILLOW                       ❸
    Last interactive login on Wednesday,  11-DEC-1995 10:20      ❹
    Last non-interactive login on Monday, 30-NOV-1995 17:39      ❺
        2 failures since last successful login                   ❻

          You have 1 new mail message.                           ❼

  $
```

Note the following about the example:

❶ The announcement message identifies the node (and, if relevant, the VMScluster name). It may also warn unauthorized users that unlawful access is prohibited. The system manager or security administrator can control both the appearance and the content of this message.

❷ A disconnected process message informs you that your process was disconnected at some time after your last successful login but is still available. You have the option of reconnecting to the old process, in the state it was in before you were disconnected.

The system displays the disconnected job message only when the following conditions exist:

- The terminal where the interruption occurred is set up as a virtual terminal.
- Your terminal is set up as one that can be disconnected.
- During a recent session, your connection to the central processing unit (CPU) through that terminal was broken before you logged out.

In general, the security administrator should allow you to reconnect to a disconnected job because this ability poses no special problems for system security. However, the security administrator can disable this function by changing the setup on terminals and by disabling virtual terminals on the system. (For information on setting up and reconnecting to virtual terminals, refer to the *OpenVMS System Manager's Manual*.)

❸ A welcome message indicates the version number of the OpenVMS operating system that is running and the name of the node on which you are logged in. The system manager can choose a different message or can suppress the message entirely.

❹ The last successful interactive login message provides the time of the last completed login for a local, dialup, or remote login. (The system does not count logins from a subprocess whose parent was one of these types.)

❺ The last successful noninteractive login message provides the time the last noninteractive (batch or network) login completed.

❻ The number of login failures message indicates the number of failed attempts at login. (An incorrect password is the only source of login failure that is counted.) To attract your attention, a bell rings after the message appears.

❼ The new mail message indicates if you have any unread mail messages.

2.4.3 Suppressing Messages

A security administrator can suppress the announcement and welcome messages, which include node names and operating system identification. Because login procedures differ according to operating system, it is more difficult to log in without this information.

The last login success and failure messages are optional. Your security administrator can enable or disable them as a group. Sites with medium-level or high-level security needs display these messages because they can indicate break-in attempts. In addition, by showing that the system is monitoring logins, these messages can be a deterrent to potential illegal users.

2.4.4 Successful Login Messages

Each time you log in, the system resets the values for the last successful login and the number of login failures. If you access your account interactively and do not specify an incorrect password in your login attempts, you may not see the last successful noninteractive login and login failure messages.

2.5 Types of Logins and Login Classes

2.5.1 Overview

Logins can be either interactive or noninteractive. When you log in interactively, you enter a user name and a password. In noninteractive logins, the system performs the identification and authentication for you; you are not prompted for a user name and password.

2.5.2 Login Classes

In addition to interactive and noninteractive logins, the OpenVMS operating system recognizes different classes of logins. How you log in to the system determines the **login class** to which you belong. Based on your login class, as well as the time of day or day of the week, the system manager controls your access to the system.

2.5.3 Interactive Logins

Interactive logins include the following login classes:

- Local

  You log in from a terminal connected directly to the central processor or from a terminal server that communicates directly with the central processor.

- Dialup

  You log in to a terminal that uses a modem and a telephone line to make a connection to the computer system. Depending on the terminal that your system uses, you might need to execute a few additional steps initially. Your site security administrator can give you the necessary details.

- Remote

  You log in to a node over the network by entering the DCL command SET HOST. For example, to access the remote node HUBBUB, you enter the following command:

  ```
  $ SET HOST HUBBUB
  ```

  If you have access to an account on node HUBBUB, you can log in to that account from your local node. You have access to the facilities on node HUBBUB, but you remain physically connected to your local node.

  For additional information on remote sessions, see Section 2.12.5.

2.5.4 Noninteractive Logins

Noninteractive logins include the following:

- Network Logins

  The system performs a network login when you initiate a network task on a remote node, such as displaying the contents of a directory or copying files stored in a directory on another node. Both your current system and the remote system must be nodes in the same network. In the file specification, you identify the target node and provide an access control string, which includes your user name and password for the remote node.

  For example, a network login occurs when user GREG, who has an account on remote node PARIS, enters the following command:

  ```
  $ DIRECTORY PARIS"GREG 8G4FR93A"::WORK2:[PUBLIC]*.*;*
  ```

  This command displays a listing of all the files in the public directory on disk WORK2. It also reveals the password 8G4FR93A. A more secure way to perform the same task would be to use a proxy account on node PARIS. For an example of a **proxy login**, see Section 19.6.5.

- Batch Logins

  The system performs a batch login when a batch job that you submitted runs. Authorization to build the job is determined at the time the job is submitted. When the system prepares to execute the job, the job controller creates a noninteractive process that logs in to your account. No password is required when the job logs in.

2.6 Login Failures

2.6.1 Common Login Failures

Logins can fail for any number of reasons. One of your passwords might have changed or your account might have expired. You might be attempting to log in over the network or from a modem but be unauthorized to do so. The following table summarizes common reasons for login failure:

Failure Indicator	Reason
No response from the terminal	A defective terminal, a terminal that requires a system password, or a terminal that is not powered on.
No response from any terminal	The system is down.

Failure Indicator	Reason
No response from the terminal when you enter the system password	The system password changed.
System messages:	
"User authorization failure"	A typing error in your user name or password. The account or password expired.
"Not authorized to log in from this source"	Your particular class of login (local, dialup, remote, interactive, batch, or network) is prohibited.
"Not authorized to log in at this time"	You do not have access to log in during this hour or this day of the week.
"User authorization failure" (and no known user failure occurred)	An apparent break-in has been attempted at the terminal using your user name, and the system has temporarily disabled all logins at that terminal by your user name.

The following sections describe the reasons for login failure in more detail.

2.6.2 Terminals That Require System Passwords

You cannot log in if the terminal you attempt to use requires a system password and you are unaware of the requirement. All attempts at logging in fail until you enter the system password.

If you know the system password, perform the steps described in Section 2.3.7. If your attempts fail, it is possible that the system password has been changed. If you do not know the system password and you suspect that this is the problem, try to log in at another terminal or request the new system password.

2.6.3 Login Class Restrictions

If you attempt a class of login that is prohibited in your UAF record, your login will fail. For example, your security administrator can restrict you from logging in over the network. If you attempt a network login, you receive a message telling you that you are not authorized to log in from this source.

Your security administrator can restrict your logins to include or exclude any of the following classes: local, remote, dialup, batch, or network.

2.6.4 Shift Restrictions

Another cause of login difficulty is failure to observe your shift restrictions. A system manager or security administrator can control access to the system based on the time of day or the day of the week. These restrictions are imposed on classes of logins. The security administrator can apply the same work-time restrictions to all classes of logins or choose to place different restrictions on different login classes.

If you attempt a login during a time prohibited for that login class, your login fails. The system notifies you that you are not authorized to log in at this time.

2.6.5 Batch Jobs During Shift Restrictions

When shift restrictions apply to batch jobs, jobs you submit that are scheduled to run outside your permitted work times are not run. The system does not automatically resubmit such jobs during your next available permitted work time. Similarly, if you have initiated any kind of job and attempt to run it beyond your permitted time periods, the job controller aborts the uncompleted job when the end of your allocated work shift is reached. This job termination behavior applies to all jobs.

2.6.6 Failures During Dialup Logins

Your security administrator can control the number of opportunities you are given to enter a correct password during a dialup login before the connection is automatically broken.

If your login fails and you have attempts remaining, press the Return key and try again. You can do this until you succeed or reach the limit. If the connection is lost, you can redial the access line and start again.

The typical reason for limiting the number of dialup login failures is to discourage unauthorized users attempting to learn passwords by trial and error. They already have the advantage of anonymity because of the dialup line. Of course, limiting the number of tries for each dialup does not necessarily stop this kind of break-in attempt. It only requires the perpetrator to redial and start another login.

2.6.7 Break-In Evasion Procedures

If anyone has made a number of failed attempts to log in at the same terminal with your user name, the system can respond as though a **break-in attempt** is in progress. That is, the system concludes that someone is attempting to gain illegal access to the system by using your user name.

At the discretion of your security administrator, break-in evasion measures can be in effect for all users of the system. The security administrator controls how many password attempts are allowed over what period of time. Once break-in evasion tactics are triggered, you cannot log in to the terminal—even with your correct password—during a defined interval. Your security administrator can tell you how long you must wait before

reattempting the login, or you can move to another terminal to attempt a login.

If you suspect that break-in evasion is preventing your login and you have not personally experienced any login failures, contact your security administrator immediately. Together, you should attempt another login and check the message that reveals the number of login failures since the last login to confirm or deny your suspicion of break-in attempts. (If your system does not normally display the login message, your security administrator can use the Authorize utility (AUTHORIZE) to examine the data in your UAF record.) With prompt action, your security administrator can locate someone attempting logins at another terminal.

2.7 Changing Passwords

2.7.1 Overview

Changing passwords on a regular basis promotes system security. To change your password, enter the DCL command SET PASSWORD.

The system manager can allow you to select a password on your own or can require that you use the automatic password generator when you change your password. If you select your own password, note that the password must follow system restrictions on length and acceptability (see Section 2.3.5).

There is no restriction on how many times you can change your password in a given period of time.

2.7.2 Example: Password Error Messages

In the following example, the password choice is too short:

```
$ SET PASSWORD
Old password:
New password:
%SET-F-INVPWDLEN, password length must be between 12 and 32
characters; password not changed
```

2.7.3 Selecting Your Own Password

If your system manager does not require use of the automatic password generator, the SET PASSWORD command prompts you to enter the new password. It then prompts you to reenter the new password for verification, as follows:

```
$ SET PASSWORD Return
New password:        Return
Verification:        Return
```

If you fail to enter the same new password twice, the password is not changed. If you succeed in these two steps, there is no notification. The command changes your password and returns you to the DCL prompt.

Even though your security administrator might not require the password generator, you are strongly encouraged to use it to promote the security of your system.

2.7.4 Using Generated Passwords

If your system security administrator decides that you must let the system generate the password for you automatically, the system provides you with a list of password choices when you enter the DCL command SET PASSWORD. (If your system is not set up to use automatically generated passwords, you can use them by specifying the SET PASSWORD command with the /GENERATE qualifier.) The character sequence resembles native language words to make it easy to remember, but it is unusual enough to be difficult for outsiders to guess. Because system-generated passwords vary in length, they become even more difficult to guess.

Note

The password generator uses basic syllabic rules to generate words but has no real knowledge of any language. As a result, it can unintentionally produce words that are offensive.

2.7.5 Example

In the following example, the system automatically generates a list of passwords made up of random sequences of characters. The minimum password length for the user in the following example has been set to 8 characters in their UAF record.

```
$ SET PASSWORD
Old password:           Return ❶

reankuna      rean-ku-na     ❷
cigtawdpau    cig-tawd-pau
adehecun      a-de-he-cun
ceebatorai    cee-ba-to-rai
arhoajabad    ar-hoa-ja-bad

Choose a password from this list, or press Return to get a new list ❸
New password:           Return ❹
Verification:           Return ❺
$ ❻
```

Note the following about the example:

❶ The user correctly specifies the old password and presses the Return key.

❷ The system responds with a list of five password choices ranging in length from 8 to 10 characters. Usually, the password that is easiest to pronounce is easiest to remember; therefore, it is the best choice.

VAX

On OpenVMS VAX systems, representations of the same word divided into syllables are displayed to the right of each password choice (as shown here).♦

❸ The system informs the user that it is possible to request a new list by pressing the Return key in response to the prompt for a new password.

❹ The user enters one of the first five possible passwords and presses the Return key.

❺ The system recognizes that this password is one provided by the automatic password generator and responds with the verification prompt. The user enters the new password again and presses Return.

❻ The system changes the password and responds with the DCL prompt.

2.7.6 Generated Passwords: Disadvantages

There are two disadvantages to using generated passwords:

- There is a possibility that you might not remember your password choice. However, if you dislike all the password choices in your list or think none are easy to remember, you can always request another list.
- There is a potential for disclosure of password choices from the display that the command produces. To protect your account, change your password in private. If you perform the change on a video terminal, clear the display of password choices from the screen after the command finishes. If you use a printing terminal, properly dispose of all hardcopy output.

  If you later realize that you failed to protect your password in these ways, change your password immediately. Depending on site policy or your own judgment concerning the length of time your account was exposed, you should notify your security administrator that a security breach could have occurred through your account.

2.7.7 Changing a Secondary Password

To change a secondary password, use the DCL command SET PASSWORD/SECONDARY. You are prompted to specify the old secondary password and the new secondary password, just as in the procedure for changing the primary password. To remove a secondary password, press the Return key when you are prompted for a new password and verification.

You can change primary and secondary passwords independently, but both are subject to the same change frequency because they share the same password lifetime.

2.7.8 Changing Passwords at Login

Even if your current password has not yet expired, you can change your password when you log in to the system by including the /NEW_PASSWORD qualifier with your user name. When you enter the /NEW_PASSWORD qualifier after your user name, the system prompts you to set a new password immediately after login.

2.7.9 Example

The following example shows how to change your password when you log in:

```
  WILLOW - A member of the Forest Cluster

Username: RWOODS/NEW_PASSWORD
Password:
         Welcome to OpenVMS on node WILLOW
            Last interactive login on Tuesday, 7-NOV-1995 10:20
            Last non-interactive login on Monday, 6-NOV-1995 14:20

Your password has expired; you must set a new password to log in
New password:
Verification:
```

2.8 Password and Account Expiration Times

2.8.1 Overview

Your system manager can set up your account so that your password, or the account itself, expires automatically on a particular date and time. Password expiration times promote system security by forcing you to change your password on a regular basis. Account expiration times help to ensure that accounts are available only for as long as they are needed.

2.8.2 Expired Passwords

As you approach the expiration time of your password, you receive an advance warning message. The message first appears 5 days before the expiration date and at each subsequent login. The message appears immediately below the new mail message and sounds the bell character on your terminal to attract your attention. The message indicates that your password is expiring, as follows:

```
WARNING -- Your password expires on Thursday 11-DEC-1995 15:00
```

If you fail to change your password before it expires, you receive the following message when you log in:

```
Your password has expired; you must set a new password to log in
New password:
```

The system prompts you for a new password or, if automatic password generation is enabled, asks you to select a new password from those listed. You can abort the login by pressing Ctrl/Y. At your next login attempt, the system again prompts you to change your password.

2.8.3 Using Secondary Passwords

If secondary passwords are in effect for your account (see Section 2.3.6), the secondary password expires at the same time as the primary one. You are prompted to change both passwords. If you change the primary password and press Ctrl/Y before changing the secondary password, the login fails. The system does not record a password change.

2.8.4 Failure to Change Passwords

If the system manager decides not to force you to change your expired password upon logging in, you receive one final warning when you log in after your password expires, as follows:

```
WARNING -- Your password has expired; update immediately with
SET PASSWORD!
```

At this point, if you do not change the password or if the system fails before you have the opportunity to do so, you will be unable to log in again. To regain access, see your system manager.

2.8.5 Expired Accounts

If you need your account for a specific purpose for a limited time only, the person who creates your account may specify a period of time after which the account lapses. For example, student accounts at universities are typically authorized for a single semester at a time.

Expired accounts deny logins automatically. You receive no advance warning message before the account expiration date, so it is important to know in advance your account duration. The account expiration resides in the UAF record, which can be accessed and displayed only through the use of the OpenVMS Authorize utility (AUTHORIZE) by users with the SYSPRV privilege or equivalent—normally, your system manager or security administrator.

When your account expires, you receive an authorization failure message at your next attempted login. If you need an extension, follow the procedures defined at your site.

2.9 Guidelines for Protecting Your Password

2.9.1 Overview

Illegal system accesses involving the use of a correct password are more often traced to disclosure of the password by its owner than to surreptitious discovery. It is vital that you do not reveal your password to anyone.

2.9.2 Guidelines

You can best protect your password by observing the following rules:

- Select reasonably long passwords that cannot be guessed easily. Avoid using words in your native language that appear in a dictionary. Consider including numbers in your password. Alternatively, let the system generate passwords for you automatically.

- Never write down your password.
- Never give your password to another user. If another user obtains your password, change it immediately.
- Do not include your password in any file, including the body of an electronic mail message. (If anyone else reveals a password to you, delete the information promptly.)

  The character strings that appear in conjunction with your actual password can make it easy for someone to find your password in a file. For example, a quotation mark followed by two colons (`"::`) always comes after a user name and password in an access control string. Someone attempting to break into the system could obtain your password by searching inadequately protected files for this string. Another way in which you might reveal your password is by using the word ‘‘password’’ in a text file, for example:

  ```
  My password is GOBBLEDYGOOK.
  ```
- If you submit a batch job on cards, do not leave your password card where others may be able to obtain your password from it.
- Do not use the same password for accounts on different systems.

  An unauthorized user can try one password on every system where you have an account. The account that first reveals the password might hold little information of interest, but another account might yield more information or more privileges, ultimately leading to a far greater security breach.
- Before you log in to a terminal that is already on, invoke the secure terminal server feature (if enabled) by pressing the Break key. This is particularly relevant when you are working in a public terminal room.

  A password grabber program is a special program that displays an empty video screen, a screen that appears to show the system has just been initialized after a crash, or a screen that shows a nonexistent logout. When you attempt to log in, the program runs through the normal login sequence so you think you are entering your user name and password in a normal manner. However, once the program receives this key information and passes it on to the perpetrator, it displays a login failure. You might think you mistyped your password and be unaware that you have just revealed it to someone else.

  To eliminate this possibility, your security administrator might advise you to press the Break key before logging in. Pressing the Break key invokes the **secure terminal server** feature for the terminal, if it has been enabled by the security administrator. The secure server ensures that the OpenVMS login program is the only program able to receive your login.

- Unless you share your password, change it every 3 to 6 months. Digital warns against sharing passwords. If you do share your password, change it every month.
- Change your password immediately if you have any reason to suspect it might have been discovered. Report such incidents to your security administrator.
- Do not leave your terminal unattended after you log in.

  You might think the system failed and came back up again, when actually someone has loaded a password-stealing program. Even a terminal that displays an apparently valid logout message might not reflect a normally logged out process.
- Check your last login messages routinely. The password-stealing program cannot actually increase the login failure count, although it looks like a login failure to you. Be alert for login failure counts that do not appear following your failure or that are one less than the number you experienced. If you observe this or any other unusual failure during a login, change your password immediately and notify your security administrator.

2.10 Recognizing System Responses

2.10.1 Overview

The system responds to the commands you enter in one or more of the following ways:

- By executing the command. Generally, you know your command has executed successfully when the system prompt returns (by default, the dollar sign).
- By executing the command and informing you in a message what it has done.
- By informing you of errors, if execution of a command is unsuccessful.
- By supplying values (defaults) you have not supplied.

2.10.2 Default Actions

A default is the value supplied by the operating system when you do not specify one yourself. For example, if you do not specify the number of copies as a qualifier for the PRINT command, the system uses the default value 1. The operating system supplies default values in several areas, including command qualifiers and parameters. The defaults that the operating system uses with specific commands are described in each command's entry in the *OpenVMS DCL Dictionary*.

2.10.3 Informational System Messages

The system responds to some commands by displaying information in a system message about what it has done. For example, when you use the PRINT command, the system displays the job identification number it assigned to the print job and shows the name of the print queue the job has entered.

```
$ PRINT MYFILE.LIS Return
     Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

Not all commands display informational messages. Successful completion of a command is usually indicated when the DCL prompt returns. Unsuccessful completion is always indicated by one or more error messages.

2.10.4 System Error Messages

If you enter a command incorrectly, the system displays a system message and prompts you for the correct command string, as the following example shows:

```
$ CAPY Return
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \CAPY\
$
```

The format for the 3-part code is:

DCL-W-IVVERB

where:

DCL — The OpenVMS facility or component name that returned the error. In this example, the message is from DCL, the default **command interpreter**.

W — A severity level that indicates a warning. Other severity levels include S (success), I (information), E (error), and F (fatal or severe error).

IVVERB — The type of message. The message can be identified by the mnemonic IVVERB in the OpenVMS system messages documentation or by using the Help Message utility (MSGHLP) described in Section 2.11.5.

2.10.5 System Error Messages During Command Execution

You can also receive system error messages during command execution if the system cannot perform the function you have requested. For example, if you type a PRINT command correctly but the file you specify does not exist, the PRINT command informs you of the error with a message like the following:

```
$ PRINT NOFILE.DAT Return
%PRINT-E-OPENIN, error opening CLASS1:[MAYMON]NOFILE.DAT; as input
-RMS-E-FNF, file not found
$
```

The first message is from the PRINT command. It tells you it cannot open the specified file. The second message indicates the reason for the first; that is, the file cannot be found. **RMS** refers to the OpenVMS file-handling software, Record Management Services; error messages related to file-handling are generally OpenVMS RMS messages.

2.10.6 Checking Your Current Process

If you suspect that your process is not doing what you think it should be doing, press Ctrl/T. Ctrl/T displays a single line of statistical information about the current process. The statistical information includes node and user name, current time, current process, **central processing unit (CPU)** usage, number of page faults, level of I/O activity, and **memory** usage, which is listed in number of CPU-specific pages.

When you press Ctrl/T during an interactive terminal session, it momentarily interrupts the current command, command procedure, or image to display statistics. Although Ctrl/T disrupts the characters on the screen, it does not affect any procedure or editing session. For example, if a user named MCCARTHY on node GREEN presses Ctrl/T while using the EVE editor, the following line is displayed in the EVE message window:

```
GREEN::MCCARTHY  13:45:02 EVE    CPU=00:00:03.33 PF=778 IO=295 MEM=315
```

To refresh the screen, press Ctrl/W.

2.10.7 Enabling Ctrl/T

Ctrl/T is disabled by default. If you know your system is running and Ctrl/T does not display statistical information, you can enable Ctrl/T with the DCL command SET CONTROL=T. Enter the command at DCL level (at the dollar sign ($) prompt), then press Ctrl/T again. Ctrl/T will remain in effect for the duration of your process, unless it is disabled from a program or command such as SET NOCONTROL=T. Note that your terminal must be set to BROADCAST mode for Ctrl/T to display on your screen. To set your terminal to BROADCAST mode, enter the DCL command SET TERMINAL/BROADCAST at the DCL prompt.

2.11 Getting Help About the System

2.11.1 Overview

When you are logged in to the operating system, you can obtain information about using the system and available commands by using the HELP command. You can also get help on system messages by entering the HELP/MESSAGE command as shown in Section 2.11.5.

2.11.2 Using Online Help

Use the following procedure to get help on OpenVMS commands and utilities:

Step	Task
1	Enter HELP at the DCL prompt and press Return. HELP displays a list of topics and the Topic? prompt.
2	To see information about one of the topics, type the topic name after the prompt and press Return.
3	If you want information on one of the subtopics, type the name after the prompt and press Return. HELP displays information about that subtopic.
4	To redisplay the SHOW USERS topic and the list of subtopics, enter a question mark (?) at the Subtopic? prompt. If you want to read all of the listed subtopics, enter an asterisk (*).
5	If you want information on another topic, press Return. Help displays the Topic? prompt.
6	To exit Help, press Return until you return to the DCL prompt.

2.11.3 Example

The following example shows the commands that you would enter to look for help about the SHOW USERS command:

```
$ HELP

HELP
.
. (HELP message text and subtopics)
.

Topic? SHOW USERS Return

SHOW

  USERS

     Displays the user name and node name (in a VAXcluster environment)
     of interactive, subprocess, and batch users on the system.

     Format

       SHOW USERS  [username]


    Additional information available:

    PARAMETER  QUALIFIER
    /BATCH     /CLUSTER   /FULL      /INTERACTIVE     /NETWORK   /NODE
    /OUTPUT    /SUBPROCESS
    Examples

SHOW USERS Subtopic? EXAMPLES

SHOW

  USERS
```

```
    Examples
.
. (SHOW USERS Examples message text and subtopics, if any)
.
SHOW USERS Subtopic? Return
SHOW Subtopic? Return
Topic? Return
$
```

2.11.4 Getting Help on Specific Commands

If you know the command you need information about, enter HELP and the command name. For example, to get help about the SHOW USERS command enter the following command:

```
$ HELP SHOW USERS Return
```

If you need help but do not know what command or system topic to specify, enter the command HELP with the word HINTS as a parameter. Each task name listed in the HINTS text is associated with a list of related command names and system information topics.

The *OpenVMS DCL Dictionary* contains more information about the HELP command.

2.11.5 Getting Help on System Messages

Use the Help Message utility (MSGHLP) to get online help for system messages. To display information on how the last command completed, type:

```
$  HELP/MESSAGE
```

You can also display information about a specific message by including the message identifier or words from the message text. For example:

```
$  HELP/MESSAGE BADACP
```

A message and its description can also be accessed by entering the message status code. For example:

```
$  HELP/MESSAGE/STATUS=%X00038090 Return
```

If you do not know the message status code, you can view it by entering the command SHOW SYMBOL followed by the $STATUS global symbol. For example:

```
$  SHOW SYMBOL $STATUS
  $STATUS == "%X00038090"
```

The Help Message utility allows you to update the messages database with your own messages or to add comments to existing message descriptions. You can also extract a subset of messages from the messages database to create and print your own customized messages documentation. For details on how to use the Help Message utility, see *OpenVMS System Messages: Companion Guide for Help Message Users*.

2.12 Logging Out of the System

2.12.1 Overview

When you finish using the system, always **log out**. This prevents unauthorized users from accessing your account and the system. It is also a wise use of system resources; the resources you no longer need are available for other users.

2.12.2 LOGOUT Command

To log out, enter LOGOUT at the DCL prompt. For example:

```
$ LOGOUT Return
```

The system displays a message, similar to the following message, confirming that you are logged out of the system:

```
$ LOGOUT Return
HARRIS logged out at 11-DEC-1995  12:42:48.12
```

2.12.3 When to Log Out

You can log out of the system only when you are at the DCL prompt ($). You cannot enter the LOGOUT command while you are compiling or executing a program, using a text editor (such as EDT or EVE), or running a utility (such as Mail). First you must exit the program, editor, or utility. When the system displays the DCL prompt, you can log out.

2.12.4 Obtaining Accounting Information

To find out how much time you spent at the terminal (elapsed time), how much computer time you used (charged CPU time), and other accounting information, enter LOGOUT/FULL at the DCL prompt. For example:

```
$ LOGOUT/FULL Return
```

The system displays information similar to the following:

```
SIMPSON logged out at 11-DEC-1995  12:42:48.12

Accounting information:
 Buffered I/O count:      8005   Peak working set size:    212
 Direct I/O count:         504   Peak virtual size:        770
 Page faults:             1476   Mounted volumes:            0
 Charged CPU time:0 00:00:50.01  Elapsed time:0 02:27:43.06
```

2.12.5 Ending a Remote Session

You can end a remote session in two ways:

- Use the remote system's logout procedure (for example, on an OpenVMS system, use the LOGOUT command).
- Press Ctrl/Y twice to obtain the host system's prompt, which asks whether you want to abort the remote session. Answer YES (Y) if you want to abort the remote session. This method works regardless of the type of system running on the remote node.

When you end a remote session, the system displays the message ''%REM-S-END, control returned to node NODENAME::'' and returns you to the process on the system from which you made the remote node connection.

2.12.6 Lost Network Connections

If the network connection to a remote system is lost, DECnet will retransmit your data in an attempt to reestablish communications. If DECnet is unable to reestablish communications within a predetermined timeout period, your connection to the remote system is terminated and the system displays the message ‘‘Path lost to partner’’.

2.13 Logging Out Without Compromising System Security

2.13.1 Protecting Your Resources

Logging out of a session conserves system resources and protects your files. Leaving a terminal on line represents one of the greatest sources of inside break-ins. When you leave your terminal on line and your office open, you have effectively given away your password and your privileges and have left your files and those of the other members of your group unprotected. Any user can easily and quickly transfer all files accessible through your account. A malicious insider could rename and delete your files and any other files to which you have write access. If you have special privileges, especially privileges in the Files or All category, a malicious user can do major damage.

Log out when you leave your office even for a brief period of time. If you have performed remote logins, you must log out of each node.

2.13.2 Reasons to Clear Your Terminal Screen

Clear your screen each time you log out of a terminal to ensure that your user name, node name, and operating system are not revealed to anyone else. If you are logging out after a remote login, the name of the node to which you return (the local node) is also revealed. If you access multiple accounts remotely over the network, the final sequence of logout commands reveals all the nodes and user names that are accessible to you on each node (excluding the name of the furthest node reached). To those who can recognize the operating system from the prompt or a logout message, these displays also reveal the operating system.

2.13.3 Clearing Your Terminal Screen

At some sites, it might be important to leave nothing but the logout message on your screen, as follows:

- If you are using a VT200 or later series terminal, you can clear the screen by pressing the Set-Up key and selecting the item from the resulting menu that corresponds to Clear Display.
- If you are using a VT100 series terminal, press the Set-Up key. Then press the key marked for reset (the 0 key) followed by the Return key.

  Alternately, to preserve temporary parameters, press the Set-Up key and then press the key marked 80/132 columns (the 9 key) twice.

After the screen clears, the cursor is positioned at the top of the screen, next to the DCL prompt. Enter the DCL command LOGOUT at the prompt. The only information remaining after you log out is your logout command and the logout completion message. For example:

```
$ LOGOUT
  RDOGWOOD     logged out at 11-DEC-1995 19:39:01.43
```

2.13.4 Disposing of Hardcopy Output

After you log out from a hardcopy terminal, remove, file, or dispose of all hardcopy output that might reveal sensitive information. Your security administrator should provide direction on preferred procedures. Many sites use paper shredders or locked receptacles for this purpose. Handle output that you plan to save just as carefully.

You should also dispose of hardcopy output if the system fails before you log out. In addition, if you will not be present when the system is initialized, turn your terminal off.

2.13.5 Breaking the Connection to a Dialup Line

Your security administrator might ask you to break the connection to a dialup line when you log out. If you anticipate no further immediate use of the line, use the LOGOUT command with the /HANGUP qualifier. The /HANGUP qualifier directs the system to automatically break the connection to the dialup line after you log out.

Note

The effectiveness of the /HANGUP qualifier depends on how your system manager configures your modem line and how the line connects to the computer. It does not work on lines connected to a terminal server.

2.13.6 Reasons to Break the Connection to Dialup Lines

Breaking the connection to a dialup line:

- Prevents others from taking advantage of an open access line. To access the line, someone must know the access number and must personally redial.
- Is especially important if the dialup line you use is in a public area or where someone might use the terminal after you.
- Saves resources by reducing the required number of dialup lines.

3

The DIGITAL Command Language: Interacting with the System

3.1 Overview

This chapter describes how to use the DIGITAL Command Language. This chapter includes information on:

- Using DCL commands
- Constructing DCL commands
- Entering DCL commands
- Rules for entering DCL commands
- Entering parameters
- Entering qualifiers
- Entering dates and times as values
- Recalling commands
- Editing the DCL command line
- Defining terminal keys
- Summary of key sequences

3.1.1 References

Complete information on all DCL commands, qualifiers, and parameters discussed in this chapter can be found in the *OpenVMS DCL Dictionary* and in online help.

3.2 Using DCL Commands

3.2.1 Definition: DIGITAL Command Language (DCL)

The DIGITAL Command Language (DCL) is a set of English-like instructions that tell the operating system to perform specific operations.

DCL commands let you do the following:

- Get information about the system
- Work with files
- Work with disks, magnetic tapes, and other devices
- Modify your work environment
- Develop and execute programs
- Provide security and ensure that resources are used efficiently

3.2.2 Entering Commands

To enter a DCL command, type the command at the DCL prompt ($) and press Return. DCL is not case sensitive; you can enter commands in either uppercase or lowercase letters.

3.2.3 Example

In the following example, the DCL command SHOW TIME is entered as follows:

```
$ SHOW TIME Return
```

The system responds by displaying the current date and time and returns the DCL prompt to indicate it is ready to accept another command:

```
11-DEC-1995  15:41:43
$
```

3.2.4 Commonly Used DCL Commands

The following table lists the DCL commands you use to perform a few common computing tasks:

Command	Task
COPY	Makes a copy of a specified file
CREATE	Creates files or directories
DELETE	Erases a specified file and removes it from a directory
DIRECTORY	Displays the contents of a directory (list of files)
EDIT	Views and changes the contents of a text file
LOGOUT	Ends your session
PRINT	Sends a specified file to a printer for printing
RENAME	Changes the name or the location of a specified file
SET	Controls how you see the system on the screen
SHOW	Displays the status of the system
TYPE	Displays the contents of a specified file on the screen

3.2.5 Key Sequences

In addition to these DCL commands, you can perform tasks by using specific key sequences. A key sequence is a shortcut or a way to get the system's attention while it is processing another command.

To enter a key sequence, hold down the Ctrl key while you press and release a second key.

3.2.6 Commonly Used Key Sequences

The following table describes a few commonly used key sequences. (Additional key sequences are listed in Section 3.12.)

Key Sequence	Function
Ctrl/C	During command entry, cancels command processing. Ctrl/C displays on your screen as Cancel.
Ctrl/Y	Interrupts command processing. Ctrl/Y displays on your screen as Interrupt.
Ctrl/T	Displays information about the current process, unless the system is temporarily unresponsive or is set to NOBROADCAST. For more information on using Ctrl/T, see Section 2.10.

3.3 Constructing DCL Commands

3.3.1 Overview

Like a spoken language, DCL is made up of *words* (vocabulary) and *word order* (syntax or format). The following sections describe these two elements and explain how to construct a valid DCL command.

3.3.2 Format of a DCL Command

The following example shows the general format and parts of a DCL command line:

```
$  PRINT/COPIES = 5  GROCERY.LIS  Return
❶  ❷     ❸     ❹     ❺     ❻
```

The following list describes each element of the DCL command line:

❶ DCL prompt

The dollar sign ($) is the default DCL prompt. When you work interactively with DCL, DCL displays the prompt when it is ready to accept a command.

❷ DCL command

A DCL command specifies the name of the command. The command can be a built-in command, a command that invokes a program, or a foreign command. In this example, the DCL command is PRINT.

❸ Qualifier

A qualifier modifies the action taken by the command. Some qualifiers modify the entire command, while others can modify specific **command parameters**. Some qualifiers can accept values. Qualifiers are always preceded by a slash (/). In this example, the qualifier is /COPIES.

❹ Value

A value modifies a qualifier and is often preceded by an equal sign (=). A value can be a file specification, a character string, a number, or a DCL keyword. A keyword is a word reserved for use in certain specified formats.

In this example, the value is 5 (for 5 copies).

❺ Parameter

A parameter specifies what the command acts upon. You must position parameters in a specified order within the command. The Examples of parameter values include file specifications, queue names, and logical names.

❻ Return key

The Return key ends the DCL command line and signals to the system that the command is ready for processing.

3.3.3 Other Components of Command Lines

The following items may also be used in a DCL command line:

- Labels

  Labels identify lines in command procedures. Use labels only within command procedures, which are described in Chapter 15 and Chapter 16.

- Keywords

  Keywords are words that are defined for use in certain specified formats. You must use keywords exactly as listed in the description of the particular DCL command you want to specify. For example, system, owner, group, and world are DCL keywords for the /PROTECTION qualifier of the SET SECURITY command. (A DCL keyword can also have a value.)

- Wildcard characters

  Wildcard characters are the asterisk (*), percent sign (%), ellipsis (...) and hyphen (-). They can be used within, or in place of, a file name, file type directory name, or version number in a file specification to indicate *all* for the given field. For information about using wildcard characters with files and directories, see Chapter 4 and Chapter 5.

3.3.4 Syntax

Just as a spoken language depends on the order of words to create meaning, DCL requires that you put the correct elements of the command line in a specific word order or format.

Following are two examples of the syntax, or format, used for typical DCL commands:

```
command/qualifier=value=keyword

command parameter/qualifier
```

3.4 Entering DCL Commands

3.4.1 Specifying Parameters

When you enter a DCL command, some parameters are required; they must be entered on the command line. If you do not enter them, the system prompts you to supply the missing information. A line beginning with an underscore (_) means that the system is waiting for your response.

When you are prompted for an optional parameter, press Return to omit it. At any prompt, after you enter the required parameter, you can enter one or more of the remaining parameters and any additional qualifiers.

Note that you must enclose in quotation marks ("") any parameter containing a slash (/) or at sign (@).

3.4.2 Example

In the following example, the TYPE command requires a file specification. Because a file specification is a required parameter of the TYPE command, if you do not enter one, the system requests it.

```
$ TYPE
_File:   WATER.TXT
```

3.4.3 Cancelling Commands

If you press Ctrl/Z after a command prompt, DCL ignores the command and redisplays the DCL prompt.

3.4.4 Using Defaults

Some items, called defaults, need not be specified on the command line. When DCL performs an operation by **default**, it assigns a command certain values or performs certain functions associated with that command even though you may not have explicitly specified those values or functions when you entered the command. In general, the values and functions are those considered typical or expected by users.

DCL supplies default values in several areas, including command parameters and qualifiers. Parameter defaults are described in the following section. Qualifier defaults are described in Section 3.7.

3.4.5 Example

If the number of copies is not specified as a qualifier for the PRINT command, DCL uses the default value 1. In the following example, the default is overridden and multiple copies of the file are printed by including the /COPIES qualifier on the PRINT command line:

```
$ PRINT/COPIES=4 MYFILE.TXT
```

3.4.6 Entering Multiple Line Commands

If you enter a command longer than one line, you can continue the command onto the next line by following this procedure:

Continue a command line onto the next line by following this procedure:

Step	Task
1	End the command line with a hyphen (-) and press Return. The system displays an underscore (_) followed by the DCL prompt ($).
2	Enter the rest of the command line after this prompt. A line beginning with an underscore means that the system is waiting for your response.

Note the following:

- You must include the appropriate spaces between command names, parameters, and so on.
- Pressing Return after the hyphen does not add a space.
- There is no restriction to the number of continued lines you can use to enter a command, as long as you do not exceed the 1024-character limit.
- You can also enter a long command line without specifying a hyphen; the system automatically wraps text to the next line. However, separating portions of the command lines with hyphens makes the command line easier to read.

3.4.7 Example

The following example shows how to enter a multiple line command:

```
$ COPY/LOG FORMAT.TXT,FIGURE.TXT,ARTWORK.TXT -
_$ SAVE.TXT
```

3.5 Rules for Entering DCL Commands

3.5.1 Case Sensitivity

Use any combination of uppercase and lowercase letters. The DCL interpreter translates lowercase letters to uppercase. Uppercase and lowercase characters in parameter and qualifier values are equivalent unless enclosed in quotation marks (‘‘ ’’).

3.5.2 Required Spaces

- Separate the command name from the first parameter with at least one blank space or a tab.
- Separate each additional parameter from the previous parameter or qualifier with at least one blank space or a tab.

3.5.3 Required Punctuation

- Begin each qualifier with a slash (/). The slash serves as a separator and need not be preceded by blank spaces or tabs.
- If a parameter or qualifier value includes a blank space or a tab, enclose the parameter or qualifier value in quotation marks.

3.5.4 Null Characters

You cannot specify null characters (<NUL>) on a DCL command line, even if you enclose the null character in quotation marks.

3.5.5 Maximum Elements

Include no more than 127 elements (parameters, qualifiers, and qualifier values) in each command line.

Each element in a command must not exceed 255 characters. The entire command must not exceed 1024 characters after all symbols and lexical functions are converted to their values. You use symbols, described in Chapter 14, to pass information to the system in an abbreviated manner. A lexical function, described in Chapter 17, obtains information from the system, including information about system processes, batch and print **queues**, and user processes, and then substitutes the result of the operation for itself.

3.5.6 Abbreviating Commands

You can abbreviate a command as long as the abbreviated name remains unique among the defined commands on a system. DCL looks only at the first four characters for uniqueness.

For greater clarity and to ensure that your command procedures are upwardly compatible, do not abbreviate commands in command procedures.

3.5.7 Example

The following commands are equal:

```
$ PRIN/COPI=2 FORMAL_ART.TXT
$ PRINT/COPIES=2 FORMAL_ART.TXT
```

3.5.8 Commands in Command Procedures

Additional rules govern the format of commands when they are used in command procedures. See Chapter 15 and Chapter 16 for more information about using commands in command procedures.

3.6 Entering Parameters

3.6.1 Common Parameters

File specifications are the most common type of parameter. DCL commands can accept input file specifications (files that are acted upon by a command) and output file specifications (files that are created by a command).

3.6.2 Rules for Specifying Parameters

The following rules apply when specifying parameters in a command line:

- Square brackets ([]) in command descriptions indicate optional items. For example, you do not have to enter a file specification in the following command:

  DIRECTORY [file-spec]

- In a command description, anything not enclosed in square brackets is required. For example, you must enter a device name in the following command:

  SHOW PRINTER device-name

- In general, precede an output file parameter with an input file parameter.
- A parameter can be one item or a series of items. If you enter a series of items, separate the items with commas (,) or plus signs (+). Any number of spaces or tab characters can precede or follow a comma or a plus sign. Note that some commands regard the plus sign as a concatenator, not as a separator.

3.6.3 Examples

1. The following example shows how to copy the input file LISTS.TXT to the output file FORMAT.TXT:

   ```
   $ COPY LISTS.TXT FORMAT.TXT
   ```

2. The following example line shows how you can enter a list of file specifications as the parameter:

   DELETE file-spec[,...]

3. The following example shows how to specify a list of parameters. Here, three files are copied to a fourth file. The three file specifications, PLUTO.TXT, SATURN.TXT, and EARTH.TXT, constitute the first parameter. PLANETS.TXT is the second parameter. Note that there are no spaces between the PLUTO.TXT, SATURN.TXT, and EARTH.TXT file specifications.

   ```
   $ COPY PLUTO.TXT,SATURN.TXT,EARTH.TXT PLANETS.TXT
   ```

3.7 Entering Qualifiers

3.7.1 Types of Qualifiers

There are three types of qualifiers:

- Command
- Positional
- Parameter

3.7.2 Abbreviating Qualifiers

You can abbreviate any qualifier name as long as the abbreviated name remains unique among all qualifier names for the same command. However, to ensure that your command procedures are upwardly compatible, do not abbreviate commands and qualifiers in command procedures.

3.7.3 Default Qualifiers

Commands have default qualifiers; you do not have to specify a qualifier unless it is different from the command default. The following sections describe types of qualifiers and qualifier defaults. The *OpenVMS DCL Dictionary* contains default information for specific commands.

3.7.4 Command Qualifiers

A command qualifier modifies a command and can appear anywhere in the command line. However, it is a good practice to place the qualifier after the command name. If you are specifying multiple qualifiers, you should place a command qualifier with other command qualifiers that follow the command name.

3.7.5 Example

In the following example, /QUEUE is a command qualifier. The files SATURN.TXT and EARTH.TXT are queued to the print queue LN03_PRINT:

```
$ PRINT/QUEUE=LN03_PRINT SATURN.TXT,EARTH.TXT
```

3.7.6 Positional Qualifiers

A positional qualifier can modify commands or parameters and has different meanings depending on where you place it in the command string. If you place a positional qualifier after the command but before the first parameter, it affects the entire command string. If you place a positional qualifier after a parameter, it affects only that parameter.

3.7.7 Example

In the following example, the first PRINT command requests two copies of the files SPRING.SUM and FALL.SUM. The second PRINT command requests two copies of the file SPRING.SUM but only one copy of FALL.SUM.

```
$ PRINT/COPIES=2 SPRING.SUM,FALL.SUM
$ PRINT SPRING.SUM/COPIES=2,FALL.SUM
```

3.7.8 Parameter Qualifiers

A parameter qualifier can be used only with certain types of parameters, such as input files and output files.

For example, the BACKUP command accepts several parameter qualifiers that apply only to input and output file specifications.

3.7.9 Example

In the following example, the /CREATED and /BEFORE qualifiers, which can be specified only with input files, select specific input files for the backup operation. The asterisk (*) is a wildcard character that replaces the file name. BACKUP selects all files with the .TXT file type that were created before December 11, 1995.

```
$ BACKUP *.TXT/CREATED/BEFORE=11-DEC-1995 NEWFILE.TXT
```

3.7.10 Conflicting Qualifiers

If you use two or more contradictory qualifiers on a command line, the right-most qualifier overrides the others.

Some commands contain conflicting qualifiers that cannot be specified in the same command line. If you use incompatible qualifiers, the command interpreter displays an error message.

3.7.11 Example

Following is an example of conflicting qualifiers. Note that the PRINT command accepts only the /COPIES=2 and the /NOBURST qualifiers because they are the right-most qualifiers in the command line:

```
$ PRINT MYFILE/COPIES=3/BURST/COPIES=2/NOBURST EARTH.TXT
```

3.7.12 Values Accepted by Qualifiers

Qualifiers can accept keywords, file specifications, character strings, and numeric values. When you enter a value for a qualifier, separate the qualifier and the value with either an equal sign (=) or a colon (:).

Some qualifier keywords require additional information. In these cases, separate the keyword from its value with a colon or an equal sign.

To specify multiple keywords that require values, enclose the list in parentheses and separate the keyword and value with either an equal sign (=) or a colon (:).

3.7.13 Examples

- Either command in this example is valid:

  ```
  $ PRINT/COPIES=3 MYFILE.DAT
  ```

  ```
  $ PRINT/COPIES:3 MYFILE.DAT
  ```

- This is an example of a qualifier that requires additional information; the keyword "PROTECTION" is separated from its value by a colon:

  ```
  $ SET SECURITY/PROTECTION:GROUP:RW MYFILE.DAT
  ```

  ```
  $ SET SECURITY/PROTECTION=GROUP=RW MYFILE.DAT
  ```

- This is an example of a qualifier that requires multiple keywords, each of which require multiple values:

  ```
  $ SET SECURITY/PROTECTION=(OWNER=RWD,GROUP=RW) myfile.dat
  ```

  ```
  $ SET SECURITY/PROTECTION=(OWNER:RWD,GROUP:RW) myfile.dat
  ```

3.8 Entering Dates and Times as Values

3.8.1 Overview

Certain commands and qualifiers (such as the PRINT/AFTER command) accept date and time values. You can specify these values in one of the following formats:

- Absolute time
- Delta time
- Combination time (combines absolute and delta time formats)

3.8.2 Absolute Time Format

Absolute time is a specific date or time of day. The format for an absolute time is as follows:

[dd-mmm-yyyy][:hh:mm:ss.cc]

The fields are as follows:

dd	Day of the month: an integer in the range 1 to 31
mmm	Month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC
yyyy	Year: an integer
hh	Hour: an integer in the range 0 to 23
mm	Minute: an integer in the range 0 to 59
ss	Second: an integer in the range 0 to 59
cc	Hundredths of a second: an integer in the range 0 to 99

3.8.3 Rules for Specifying Absolute Time Format

The following rules apply when specifying absolute time:

- You can truncate the date or the time on the right.
- If you specify both a date and a time, include a colon between them.
- The date must contain at least one hyphen.
- You can omit any of the fields within the date and time as long as you include the punctuation marks that separate the fields.
- A truncated or omitted date field defaults to the corresponding fields for the current date.
- A truncated or omitted time field defaults to zero.
- If you specify a past time in a command that expects the current or a future time, the current time is used.

3.8.4 Absolute Time Keywords

You can also specify an absolute time as one of the following keywords:

TODAY	The current day, month, and year at 00:00:00.0 o'clock
TOMORROW	00:00:00.00 o'clock tomorrow
YESTERDAY	00:00:00.00 o'clock yesterday

3.8.5 Examples: Absolute Time

The following table shows examples of absolute time specifications:

Time Specification	Result
11-DEC-1995:13	1 P.M. on December 11, 1995
11-DEC	Midnight at the beginning of December 11 this year
15:30	3:30 P.M. today
19--	Midnight on the 19th day of the current year and month
19--:30	12:30 A.M. on the 19th of this month

3.8.6 Delta Time Format

Delta time is an offset (a time interval) from the current date and time to a time in the future. The general format of a delta time is as follows:

"+[dddd-][hh:mm:ss.cc]"

The fields are as follows:

dddd	Number of days; an integer in the range 0 to 9999
hh	Number of hours; an integer in the range 0 to 23
mm	Number of minutes; an integer in the range 0 to 59
ss	Number of seconds; an integer in the range 0 to 59
cc	Number of hundredths of seconds; an integer in the range 0 to 99

If a qualifier is described as a value that can be expressed as an absolute time, a delta time, or a combination of the two, you must specify a delta time as if it were part of a combination time. For example, to specify a delta time value of five minutes from the current time, use ''+:5'' (not ''0-0:5'').

3.8.7 Rules for Specifying Delta Time Format

The following rules apply when specifying delta time:

- You can truncate a delta time on the right.
- If you specify the number of days, include a hyphen.
- You can omit fields within the time as long as you include the punctuation that separates the fields.

- If you omit the time field, the default is zero.

3.8.8 Examples

The following table shows some examples of delta time specifications:

Time Specification	Result
‘‘+3-’’	3 days from now (72 hours)
‘‘+3’’	3 hours from now
‘‘+:30’’	30 minutes from now
‘‘+3-:30’’	3 days and 30 minutes from now
‘‘+15:30’’	15 hours and 30 minutes from now

3.8.9 Combination Time Format

To combine absolute and delta times, specify an absolute time plus or minus a delta time. Use one of the following formats:

"[absolute time][+delta time]"

[absolute time][-delta time]

The variable fields and default fields for absolute and delta time values are the same as those described in the preceding sections.

3.8.10 Rules for Specifying Combination Time

The following rules apply when specifying combination time:

- Precede the delta time value by a plus or minus sign. (Note that the minus sign is the same keyboard key as the hyphen.)
- Enclose the entire time specification in quotation marks if a plus or minus sign precedes the delta time value.
- Omit the absolute time value if you want to offset the delta time from the current date and time.
- Specify date and time information as completely as possible.

3.8.11 Examples

The following table shows some examples of combination time specifications:

Time Specification	Result
‘‘+5’’	5 hours from now.
‘‘-1’’	Current time minus 1 hour. The minus sign (-) indicates a negative offset. (The 1 is interpreted as an hour, not a day, because it is not followed by a hyphen.)
‘‘+:5’’	5 minutes from now.
‘‘-:5’’	Current time minus 5 minutes.

Time Specification	Result
‘‘-1-00’’	Current time minus 1 day. The minus sign (-) indicates a negative offset. The hyphen (-) separates the day from the time field.
‘‘31-DEC:+:5’’	12:05 A.M. on December 31 of the current year. The absolute time specification (before the colon) defaults to midnight on December 31 of the current year. The plus sign (+) indicates a positive offset.
31-DEC:-00:10	11:50 P.M. on December 30 of the current year. The absolute time specification (before the colon) defaults to midnight on December 31 of the current year. The minus sign (-) after DEC: indicates a negative offset.

3.9 Recalling Commands

3.9.1 Overview

At the DCL prompt, you can recall previously typed command lines to avoid retyping long command lines. Once a command is displayed, you can reexecute or edit it.

VAX

On OpenVMS VAX systems, the recall buffer holds up to 20 previously entered commands.♦

Alpha

On OpenVMS Alpha systems, the recall buffer holds up to 254 previously entered commands.♦

You can display your previously entered commands by using one of the following methods:

- Pressing Ctrl/B
- Using up arrow and down arrow keys
- Entering the RECALL command

3.9.2 Using Ctrl/B

Pressing Ctrl/B once recalls the previous command line. Pressing Ctrl/B again recalls the line before the previous line and so on to the last saved command line.

3.9.3 Using Arrow Keys

Pressing the up arrow and down arrow keys recalls the previous and successive command, respectively. Press the arrow keys repeatedly to move through the commands.

3.9.4 Using the RECALL Command

To examine previously typed command lines, type RECALL /ALL. After reviewing the available commands, you can recall a particular command line by typing RECALL and the number of the desired command.

You can also follow RECALL with the first characters of the command line you want to display. RECALL scans the previous command lines (beginning with the most recent one) and returns the first command line that begins with the characters you typed.

3.9.5 Examples

1. This is a sample display generated by typing RECALL/ALL:

```
$ RECALL/ALL

 1 SET DEFAULT DISK2:[MARSHALL]
 2 EDIT ACCOUNTS.COM
 3 PURGE ACCOUNTS.COM
 4 DIRECTORY/FULL ACCOUNTS.COM
 5 COPY ACCOUNTS.COM [.ACCOUNTS]*
 6 SET DEFAULT [.ACCOUNTS]
```

2. The following example shows how to recall the fourth command line:

```
$ RECALL 4
```

   After you press Return, the system displays the fourth command in the list at the DCL prompt. (The RECALL command itself is not placed in the buffer.)

3. The following example shows how to recall a previously entered command, EDIT ACCOUNTS.COM:

```
$ RECALL E
```

   After you press Return, the system displays the following command line:

```
$ EDIT ACCOUNTS.COM
```

3.9.6 OpenVMS Screen Management Software

If you are running a utility or an application program that uses OpenVMS screen management software, you can use Ctrl/B and the up arrow and down arrow keys to perform command recall; however, line editing must be enabled. Some utilities that have this feature are Mail, OpenVMS Debugger, Show Cluster, the System Dump Analyzer (SDA), and the EVE editor.

3.9.7 Erasing the Recall Buffer

To erase the contents of the recall buffer, enter the RECALL command with the ERASE qualifier. For example:

```
$ RECALL/ERASE
```

For security reasons, it is good practice to erase the contents of the recall buffer after you have entered commands that include passwords.

3.10 Editing the DCL Command Line

3.10.1 Overview

At the DCL command level, you can use many individual keys and key sequences to change what you type. Although different types of terminals have different operating characteristics, most have standard function keys and keys that can be used with **line editors**.

3.10.2 SHOW TERMINAL Command

To see whether line editing is enabled on your terminal, enter the SHOW TERMINAL command. The current status of line editing is displayed in the first column under Terminal Characteristics.

3.10.3 Example

In the following example, line editing is not enabled:

```
$ SHOW TERMINAL

Terminal: _VTA130:    Device_Type: VT200_Series  Owner: ROHBA
LAT Server/Port: L121/Port_3
Physical terminal: _LTA130:
   Input:   9600      LFfill:  0      Width:  80      Parity: None
   Output:  9600      CRfill:         Page:   24
Terminal Characteristics:
 Interactive        Echo               Type_ahead       No Escape
 No Hostsync        TTsync             Lowercase        Tab
 Wrap               Scope              No Remote        No Eightbit
 Broadcast          No Readsync        No Form          Fulldup
 No Modem           No Local_echo      No Autobaud      Hangup
 No Brdcstmbx       No DMA             No Altypeahd     Set_speed
 No Line Editing    Insert editing     No Fallback      No Dialup
 No Secure server   Disconnect         No Pasthru       No Syspassword
 No SIXEL Graphics  No Soft Characters No Printer Port  Numeric Keypad
 ANSI_CRT           No Regis           No Block_mode    Advanced_video
 No Edit_mode       DEC_CRT            No DEC_CRT2
```

3.10.4 SET TERMINAL Command

You can use the SET TERMINAL command to alter the way in which your terminal edits a DCL command line. By default, changes made with the SET TERMINAL command apply only to the current session. To set the terminal each time you log in, you can include SET TERMINAL commands in your LOGIN.COM file.

3.10.5 Enabling Line Editing

To enable line editing, enter the SET TERMINAL/LINE_EDIT command:

```
$  SET TERMINAL/LINE_EDIT
```

3.10.6 Changing Edit Modes

You can edit a command line in either insert or overstrike mode. In insert mode, the character you type is inserted to the left of the **cursor**. In overstrike mode, the character you type overwrites the character indicated by the cursor.

To change editing modes for a single command line, press Ctrl/A (Ctrl/A acts as a toggle). To change edit modes for your session, enter either the SET TERMINAL/INSERT or SET TERMINAL /OVERSTRIKE command.

3.10.7 Making Command Lines Wrap

If you use the SET TERMINAL/WRAP command, when you enter more characters than will fit on one line of the terminal screen, the text wraps to the next line.

You can edit only the line where your cursor appears. When text wraps, you cannot use the up arrow key to move the cursor up to edit the previous line. To move the cursor up to the previous line, use the Delete key and delete all the characters in the current line.

3.10.8 Deleting Portions of the Command Line

The Delete key on your keyboard is marked with either the word Rubout, the word Delete, or an X in a left-pointing arrow, depending on the type of terminal you are using. The Delete key back spaces over the most recently entered character and deletes it. On a **hardcopy terminal**, the deleted letters are displayed between backslash characters so you can see what is being deleted. On a video display terminal, pressing the Delete key erases the character from the screen and moves the cursor backwards.

In contrast, the Backspace key (or the left arrow key) back spaces over characters but does not delete them.

If line editing is enabled, you can use Ctrl/U to delete characters from the beginning of the line to the current cursor position. If line editing is not enabled, you can use Ctrl/U to cancel an entire line. The system ignores the line and redisplays the DCL prompt.

3.11 Defining Terminal Keys

3.11.1 Key Definitions

A key definition is a string of characters that you assign to a particular terminal key. When a key is defined, you can press it instead of typing the string of characters. A key definition usually contains all or part of a command line. Using key definitions, you can customize your keyboard so that you can enter DCL commands with fewer keystrokes. When you press a defined key, the system either displays the command on your terminal or executes the command, depending on whether the command was defined using the /TERMINATE qualifier.

3.11.2 Definable Keys

Some definable keys are automatically enabled for definition (for example, keys PF1 to PF4 and keys F17 to F20 on LK201 keyboards). However, before you can define other keys, including KP0 (keypad 0) to KP9 and the keypad keys period, comma, minus, and Enter, you must enable them for definition by entering either the SET TERMINAL/APPLICATION_KEYPAD or the SET TERMINAL/NONUMERIC command.

3.12 Summary of Key Sequences

3.12.1 Keys That Enter DCL Commands

The following keys enter DCL commands:

- Ctrl/Z and F10

  Signals the end of the file for data entered from the terminal. Ctrl/Z displays as Exit. This key is available only on an LK201 keyboard.

- Return

  Sends the current line to the system for processing. On some terminals, the Return key is labeled CR. If you are not already logged in, Return initiates a login sequence.

3.12.2 Keys That Interrupt DCL Commands

The following keys interrupt DCL commands:

- Ctrl/C and F6

  During command entry, cancels command processing. Ctrl/C displays as Cancel. This key is available only on an LK201 keyboard.

- Ctrl/T

  Momentarily interrupts terminal output to display a line of statistical information about the current process. This display includes your node and user name, the time, the name of the image you are running, and information about system resources used during your current terminal session.

  You can also use Ctrl/T to determine whether the system is operating. Ctrl/T does not return information if the system is temporarily unresponsive or if your terminal is set to NOBROADCAST. To use Ctrl/T, you must first enter the SET CONTROL=T command (in the system login command procedure, in your personal login command procedure, or interactively).

- Ctrl/Y

  Interrupts command processing. Ctrl/Y displays as Interrupt. You can disable Ctrl/Y with the command SET NOCONTROL=Y.

  Under most conditions, Ctrl/Y returns you to the DCL prompt. The program running is still active. You can enter any built-in command then continue the program with the CONTINUE command. (Press Ctrl/W to refresh the screen after you enter the CONTINUE command.)

3.12.3 Keys That Recall Commands

The following keys recall commands:

- Ctrl/B or up arrow

  Recalls up to 20 (VAX) or 254 (Alpha) previously entered commands.

- Down arrow

  Displays the next line in the recall buffer.

3.12.4 Keys That Control Cursor Position

The following keys control cursor position:

- , Delete

  Deletes the last character entered at the terminal. On some terminals, the Delete key is labeled RUBOUT. The Delete key also works when line editing is disabled.)

- Ctrl/A and F14

  Switches between overstrike and insert mode. The default mode (as set with the SET TERMINAL/LINE_EDITING command) is reset at the beginning of each line. This key is available only on an LK201 keyboard.

- Ctrl/D and left arrow

  Moves the cursor one character to the left.

- Ctrl/E

  Moves the cursor to the end of the line.

- Ctrl/F and right arrow

  Moves the cursor one character to the right.

- Ctrl/H, Backspace, and F12

  Moves the cursor to the beginning of the line. This key is available only on an LK201 keyboard.

- Ctrl/I and Tab

  Moves the cursor to the next tab stop on the terminal. The system provides tab stops at every eighth character position on a line. Tab settings are hardware terminal characteristics that, in general, you can modify. The Tab key also works when line editing is disabled.

- Ctrl/J, Linefeed, and F13

  Deletes the word to the left of the cursor. This key is available only on an LK201 keyboard.

- Ctrl/K

  Advances the current line to the next vertical tab stop.

- Ctrl/L

  Causes the cursor to go to the beginning of the next page. This use of Ctrl/L is ignored when line editing is enabled.

- Ctrl/R

  Repeats the current command line and leaves the cursor positioned where it was when you pressed Ctrl/R.

- Ctrl/U

  Deletes all text in the current input line that is to the left of the cursor.

- Ctrl/V

  Turns off some of the line editing function keys. For example, if you press Ctrl/V followed by Ctrl/D, a Ctrl/D is generated instead of the cursor moving left one character. Ctrl/D is a line terminator at DCL level.

  When combined with Ctrl/V, characters that are not line terminators have no effect. Examples are Ctrl/H and Ctrl/J. However, certain control keys, such as Ctrl/U, retain their line editing functions.

- Ctrl/X

  Cancels the current line and deletes data in the type-ahead buffer.

- F7, F8, F9, F11

  Reserved by Digital.

3.12.5 Keys That Control Screen Display

The following keys control screen display:

- Ctrl/O

  Alternately suspends and continues display of output to the terminal. Ctrl/O displays as Output off and Output on.

- Ctrl/S

  Suspends terminal output until Ctrl/Q is pressed.

- Ctrl/Q

  Resumes terminal output suspended by Ctrl/S.

- Hold Screen and No Scroll

  Suspends terminal output until the key is pressed again. The Hold Screen key is available only on an LK201 keyboard, and the No Scroll key is available only on a VT100 keyboard.

4
Files: Storing Information

4.1 Overview

This chapter describes how to create and manipulate files locally, and over a DECnet for OpenVMS network. This chapter includes information on:

- Understanding file names and file specifications
- Using wildcards with file names
- Other file names
- Creating and modifying files
- Displaying the contents of files
- Deleting files
- Protecting files from other users
- Printing files

4.1.1 References

- Additional information on all commands discussed in this chapter can be found in the *OpenVMS DCL Dictionary* and in online help.
- Additional information on Accessing Remote Nodes can be found in the *OpenVMS System Manager's Manual*.
- Information on Networks can be found in the *DECnet for OpenVMS Networking Manual*.

4.2 Understanding File Names and File Specifications

4.2.1 Overview

A file is a unit that the OpenVMS operating system uses to store human-readable and machine-readable data. When you create or name a file, you must specify certain information so that the system can locate and identify the file. You do not have to include all the elements of a complete file specification (see Section 4.2.2). However, you must include a file name or file type to identify it to both the system and you. For example, PAYROLL_MEMO.TXT is a valid file specification. PAYROLL_MEMO is the file name, and .TXT is the file type.

4.2.2 Providing a Complete File Specification

To override system defaults or to perform file operations over a network, you must provide a complete file specification. A complete file specification has the following format:

node::device:[directory]filename.filetype;version

The elements are as follows:

Node	A network node name; applicable only to systems that support DECnet for OpenVMS. Does not apply to files stored on magnetic tape.
Device	The name of the physical device on which the file is stored or is to be written. For information on accessing files stored on physical devices, see Section 12.7.
Directory	The name of the directory under which the file is cataloged. Square brackets ([ ]) or angle brackets (<>) can be used to delimit directory names. Does not apply to files stored on magnetic tape.
Filename	The name of the file. It can have up to 39 alphanumeric characters, including the hyphen and the underscore.
Filetype	Identification of the structure or the type of data in the file. The file type can have up to 39 alphanumeric characters, including the hyphen and the underscore.
Version	The version number of the file. Versions are identified by a decimal number, which is incremented by 1 each time a new version of the file is created. The system automatically assigns a version number unless you specify one.

4.2.3 Rules for File Specifications

Use the following rules to specify the elements of a file specification:

- Give the file a name that is meaningful to you. The file name can be up to 39 characters chosen from the letters A to Z (uppercase or lowercase), the numbers 0 to 9, underscores (_), hyphens (-), and dollar signs ($).
- Do not use a hyphen as the first or last character in the file name. While it is possible under some conditions to successfully create a file with the hyphen as the first character in the file name, special handling is required to access this file.
- Include 0 to 39 characters in a file type.
- Precede a file type by a period.
- Precede version numbers with a semicolon or a period. (When the system displays file specifications, it displays a semicolon in front of the file version number.)

- Do not use a directory field to refer to files on magnetic tape (directories apply only to files on disks).
- Include a node name only if your system is part of a network.
- When naming files on mass storage devices (such as disks and magnetic tapes), include the fields for file name, type, and version.

4.2.4 Default File Types Used by DCL Commands

With certain commands, if you omit the file type, the system applies a default value. The following table lists some of the more common default file types used by DCL commands:

File Type	Contents
.CLD	Command description file
.COM	Command procedure file
.DAT	Data file
.DIF	Output file created by the DIFFERENCES command
.DIR	Directory file
.DIS	Distribution list file for the Mail utility
.EDT	Startup command file for the EDT editor
.EXE	Executable program image file created by the linker
.HLB	Help text library file
.HLP	Input source file for help libraries
.INI	Initialization file
.JOU	Journal file created by the EDT editor
.LIS	Listing file created by a language compiler or assembler; default input file for the PRINT and TYPE commands
.LOG	Batch job output file
.MAI	Mail message file
.MEM	Output file created by DIGITAL Standard Runoff (DSR)
.PS	POSTSCRIPT format file
.REGIS	Regis format file
.RNO	Input source file for DIGITAL Standard Runoff (DSR)
.SIX	Sixel graphic file
.SYS	System image file

File Type	Contents
.TJL	Journal file created by the DECTPU and ACL editors
.TLB	Text library file
.TMP	Temporary file
.TPU	Command file for the EVE editor
.TPU$JOURNAL	Journal file created by the EVE editor
.TXT	Input file for text libraries or Mail utility output files

4.2.5 Default File Types for Language Source Programs

The following table lists the default file types for some high-level language source programs:

File Type	Contents
.ADA	Input source file for the DEC Ada compiler
.BAS	Input source file for the BASIC compiler
.B32	Input source file for the VAX BLISS-32 compiler
.C	Input source file for the DEC C compiler
.COB	Input source file for the VAX COBOL compiler on OpenVMS VAX systems and the DEC COBOL compiler on OpenVMS Alpha systems
.FOR	Input source file for DEC Fortran (DEC Fortran for OpenVMS VAX systems was formerly VAX Fortran)
.M64	Input source file for the MACRO-64 assembler for OpenVMS Alpha
.MAP	Memory allocation map created by the Linker utility
.MAR	Input source file for the VAX MACRO assembler or the MACRO-32 Compiler for OpenVMS Alpha.
.MLB	Macro library for the MACRO assembler
.MSG	Source file that specifies the text of messages
.OBJ	Object file created by a language compiler or assembler
.OLB	Object module library
.OPT	Options file for input to the LINK command

File Type	Contents
.PAS	Input source file for the Pascal compiler
.PLI	Input source file for the PL/I compiler
.STB	Symbol table file created by the Linker utility
.UPD	Update file of changes for a VAX MACRO source program; also input to the SUMSLP utility

4.2.6 File Versions

In addition to a file name and file type, every file has a version number. Version numbers are decimal numbers from 1 to 32,767 that differentiate versions of a file. When you create a file, the system assigns it the version number 1.

You can have several versions of the same file. Unless you specify a version number, the system uses the highest existing version number of that file. If you specify the version number 0, the system uses the highest existing version. When you modify a file with a command, application, or text editor (such as EVE) that creates a new version of the file, the file name remains the same but the version number is incremented by one.

Precede version numbers with a semicolon or a period. When the system displays file specifications, it displays a semicolon in front of the file version number.

4.2.7 Specifying File Versions

You can refer to versions of a file in a relative manner by specifying a zero or a negative version number. Specifying zero locates the latest (highest numbered) version of the file. Specifying −1 locates the next-most-recent version, −2 the version before that, and so on. To locate the earliest (lowest numbered) version of a file, specify −0 as the version number. Note that you cannot create files with a version number higher than 32767. If you attempt to create a new file with a version number higher than 32767, you will receive an error message.

4.2.8 Controlling the Number of File Versions

The /VERSION_LIMIT qualifier for the CREATE/DIRECTORY, SET DIRECTORY, and SET FILE commands lets you control the number of versions of a file. If you exceed the version limit, the system automatically purges the lowest version file in excess of the limit. For example, if the version limit is 5 and you create the sixth version of a file (ACCOUNTS.DAT;6), the system deletes the first version of the file (ACCOUNTS.DAT;1). To view the version limit on a file, enter the DIRECTORY/FULL command. The version limit is listed the File attributes section.

4.2.9 Network Node Names

A node is an individual computing system that is part of a computer network. If your system is part of a network, the node that you access when you log in is your local node. Other nodes in the network are remote nodes. Use a node name when you want to specify a file on a remote node.

4.2.10 Node Specification Format and Rules

A **node specification** has the following format:

```
node["access-control-string"]::
```

Observe the following rules when entering a node name as part of a file specification:

- Node names can contain 1 to 6 alphanumeric characters and must contain at least one alphabetic character. For example:

  ```
  AFTP1
  F2OTR2
  MYNODE
  ```

- A node name (with or without an access control string) must always be followed by a double colon (::).
- When you specify a node name, you can include a 0- to 42-character **access control string**. An access control string contains login information to be sent to the remote node. For more information on access control strings, see Section 4.2.22.

  Note that the required double colon follows the access control string.
- You can use a logical name in place of the node name. For information on logical node names, see Chapter 13.

4.2.11 Specifying Node Full Names

On OpenVMS systems, you can specify node **full names**. However, you must have **DECnet/OSI** software installed for full node names to be recognized.

Valid full node names can contain up to 255 characters and can include any characters except the following:

- Spaces
- Tabs
- The characters: comma (,), quotation marks (‘‘ ’’), slash (/), exclamation point (!), plus sign (+), at sign (@), apostrophe (’), parentheses (( )), and double colons (::)
- A single colon (:) as the first or last character

If a full node name is enclosed in quotation marks (‘‘ ’’), it can contain any characters except unmatched quotation marks. Note that if there are quotation marks within the node name, the quotation marks must be doubled and the entire string, including the quotation marks, must also be enclosed in quotation marks.

Although the OpenVMS software enforces few rules on the syntax of node names, the actual set of valid node names is constrained by the DECnet software running on your system. For further information on full names, refer to the DECnet/OSI documentation.

4.2.12 Examples

In the following example, the entire string is in quotation marks because there are quotation marks in the node name:

```
"MARY:.UNIVERSITY.""SCIENCE LAB"""
```

Other examples of valid full node names are:

```
MYNODE
MASSACHUSETTS:.BUSINESS.YOURNODE
A.B;C
```

4.2.13 Accessing Files on Remote Nodes

When you access a file on a remote node, DECnet logs in at the remote node. To do this, the system needs login information for that node. You can supply the system with an access control string. If you omit the access control string, the login information sent to the remote node is determined as follows:

- If a proxy login account exists for you on the remote node, then the system logs you in using that account. A proxy login account allows selected users to log in to a node.
- If a proxy login account does not exist, the system uses the default DECnet account for that node as specified by the local system manager.

If you include an access control string, the system uses it to log you in to the remote node. The remainder of the file specification is passed to the remote node and is interpreted there.

If you specify a local node as part of a file specification, the system logs you in over the network to perform the file operation, even though the file exists on your local node. For information about additional ways to access remote systems, see the *OpenVMS System Manager's Manual*.

4.2.14 Note: Examples in This Chapter

Throughout the remainder of this chapter, examples that specify a node name do not always include an access control string. This is because proxy accounts enable users to perform operations on the remote systems in these examples.

4.2.15 Using Network File Specifications

There are three formats for network file specifications:

- Conventional
- Foreign
- Task

In each format, the node specification can include an access control string. For more information, see the *DECnet for OpenVMS Networking Manual*.

4.2.16 File Name Format

The conventional format for files is:

node::device:[directory]filename.type;version

4.2.17 Foreign File Specification

A **foreign file specification** is a file that does not conform to OpenVMS syntax. The format used to provide a foreign file specification is:

node::"foreign-file-spec-string"

4.2.18 Example

This file name contains a question mark (?), which is not recognized as a valid file name character. Therefore, the file name must be enclosed in quotation marks (‘‘ ’’). It must also be in a format that is recognized by the operating system of the remote node you are accessing:

```
$ COPY BOSTON::"TEST?.DAT" *
```

4.2.19 Task Specification Strings

A task specification string identifies a program to be executed on the remote node. You can use task specification strings within a program to enable the program to communicate with another program on a remote node. The format used to indicate a task specification string is:

node::“task-spec-string”

4.2.20 Example

This specification identifies the program TEST2 on the remote node BOSTON:

```
BOSTON::"TASK=TEST2"
```

4.2.21 Note: ULTRIX Restrictions

There are some restrictions when you copy files to or from an ULTRIX system. For more information, see the *OpenVMS Record Management Utilities Reference Manual*.

4.2.22 Access Control String Format

Access control strings designate accounts that you can log in to on remote nodes. Node names with access control strings have the following format:

node"access-control-string"::

Enclose the access control string in quotation marks (‘‘ ’’) and follow it with a double colon (::).

On OpenVMS systems, the access control string consists of a user name, followed by one or more spaces or tabs and a password. For additional information on access control strings, see Chapter 19.

4.2.23 Example

In the following example, BOSTON is the network node name. "HIGGINS ETUHCARAP" is an access control string where:

- HIGGINS is a user name on the node BOSTON.
- ETUHCARAP is the password associated with that name:

```
$ DIR BOSTON"HIGGINS ETUHCARAP"::WEASEL2:[BORIS]ACCOUNTS.DAT
```

4.3 Using Wildcards with File Names

4.3.1 Overview

Use wildcard characters to apply a DCL command to multiple files rather than to one file at a time. The command applies to all files that match the portion of the file specification entered.

Many examples in this chapter show the use of wildcard characters in file operations. The use of wildcard characters in DCL commands varies with the individual command.

4.3.2 Types of Wildcards

There are two wildcards available for use with many DCL commands: asterisks (*) and percent signs (%). They can be used as wildcard characters in directory names, file names, and file types. (Refer to Section 5.6 for information on wildcards used with directories.) You can also use an asterisk, but not a percent sign in version numbers.

4.3.3 The Asterisk (*) Wildcard Character

Use the asterisk (*) wildcard character to match the following:

- An entire field (or a portion of it) in the directory, file name, and file type fields
- The entire version number field, but not a portion of it

You can use the asterisk (*) wildcard character as follows:

- To manipulate large numbers of files without naming them individually
- To limit the files selected to a more specific group
- In directory specifications

4.3.4 Examples

1. In the following example, the file specification selects all versions of all files in the [FROGMAN] directory:

   ```
   $ PRINT [FROGMAN]*.*;*
   ```

2. In the following example, only those files in the current default directory with the file type .DAT are displayed:

   ```
   $ TYPE *.DAT;*
   ```

3. The command in this example selects all files with the file type .DAT that exist in subdirectories one level below [FROGMAN]:

   ```
   $ DIRECTORY [FROGMAN.*]*.DAT
   ```

4. In the following example, the wildcard characters appear in the directory specification:

   ```
   $ TYPE [*.*.*]AVERAGE.*;*
   ```

   This file specification selects all versions of all files named AVERAGE with any file type that exist in any second-level subdirectory on the current default disk. For example, this file specification selects [A.B.C]AVERAGE.DAT but not [X.Y]AVERAGE.DAT.

4.3.5 The Percent Sign Wildcard Character

Use the percent sign (%) wildcard character as a substitute for any single character in a file specification. You can use the percent sign in the directory, file name, and file type fields. You cannot, however, use the percent sign in the version number field or in ANSI magnetic tape file specifications. The percent sign replaces one character position in a field, but there must be a character to replace.

You can specify the percent sign as many times as necessary and in combination with other wildcard characters.

4.3.6 Examples

- The following example displays the latest versions of all .DAT files whose names are DISTRICT followed by a single character:

  ```
  $ TYPE [JONES.TAXES.PROPERTY]DISTRICT%.DAT
  ```

  This display would include the files DISTRICT1.DAT, DISTRICT2.DAT, and DISTRICT3.DAT. The file DISTRICT4_5.DAT would not be displayed because it has more than one character after DISTRICT, nor would the file DISTRICT.DAT be displayed.

- The file specification in this example is valid:

  ```
  $ [MA*]INS%%%A*.J*;*
  ```

4.4 Other File Names

4.4.1 Null File Names and File Types

The file name and file type fields can be null. For example, the following are valid file specifications:

.TMP (file name is null)

TEMP. (file type is null)

When you specify a file in a DCL command, be careful to omit the period following a file name if the command uses a default file type.

4.4.2 Example

Because the FORTRAN command uses the default file type .FOR, the commands in this example produce different results:

```
$ FORTRAN TEMP
$ FORTRAN TEMP.
```

In the first example, the FORTRAN compiler looks for a file named TEMP.FOR because the file type is omitted. In the second example, the compiler looks for a file named TEMP. because a period following the file name indicates a null file type.

4.4.3 Alternate File Names for Magnetic Tapes

In addition to standard file names, the operating system supports an alternate file-naming convention for ANSI-labeled magnetic tapes. The format is as follows:

"filename".;version

The file name can contain 1 to 17 characters from the ASCII "a" character set. This set of characters includes numeric characters, uppercase letters, and a space, as well as the following characters:

! " % ' ( ) * + , - . / : ; < => ? & _

In addition, asterisk (*) wildcards are allowed in ANSI file names.

4.5 Creating and Modifying Files

4.5.1 Text Editors

The most versatile interactive tool for creating and modifying text files is the interactive text editor. EVE and EDT are two text editors that are included in the OpenVMS operating system; other text editors may also be available on your system.

4.5.2 Commands That Create Files

You can also create and modify files by using the DCL commands CREATE, COPY, and RENAME. The following sections describe how to create and modify files using these commands.

4.5.3 Creating Files

The CREATE command creates a text file. You cannot modify a file with the CREATE command; after you have pressed Return, you cannot return to a previous line to modify a word. You must use a text editor such as EDT or EVE to modify a file created with the CREATE command. Pressing Ctrl/Z signals the end of the file and returns you to DCL command level.

4.5.4 Example

In the following example, a file named POUND.LIS is created by entering the CREATE command and then typing lines of text:

```
$ CREATE POUND.LIS
Tag #23, Elmer Doolittle, notified
Tag #37, James Watson, notified
No tag, light brown, 30 lbs., looks part beagle
Ctrl/Z
```

4.5.5 Copying Files

You can use the COPY command to duplicate:

- The contents of an existing file in a new file
- Many files at a time
- Only those files that meet specified criterion by using the /SINCE qualifier with the COPY command

4.5.6 Examples

- In the following example, the file FEES.DAT is copied to RECORDS.DAT:

```
$ COPY FEES.DAT RECORDS.DAT
```

- In the following example, all .TXT files in the default directory are copied to another directory:

```
$ COPY *.TXT;* [SAVETEXT]*.*;*
```

- In the following example, only those files in the directory [JONES.LICENSES.DOG] that have been modified since December 11, 1995 are copied to the default directory:

```
$ COPY/SINCE=11-DEC-1995/MODIFIED [JONES.LICENSES.DOG]*.* *
```

4.5.7 File Concatenation

The COPY command can also be used to concatenate files. For example, to append FEES1.DAT to FEES.DAT (forming a new version of FEES.DAT) in your default directory, enter the following command:

```
$ COPY FEES.DAT,FEES1.DAT FEES.DAT
```

Note that there is no space between the comma after FEES.DAT and the file name FEES1.DAT.

4.5.8 Copying Files from a Remote Node to Your Node

Use the COPY command to copy files from another node to your node. For example, to copy the latest version of all files in DISK2:[PUBLIC] on node CHAOS to files with the same names in your default directory, enter the following command:

```
$ COPY CHAOS::DISK2:[PUBLIC]*.*  *
```

4.5.9 Copying Files from Your Node to a Remote Node

Use the COPY command to copy files from your node to another node. If you receive a protection violation or DECnet error message when you attempt to copy a file across systems, you can either use mail to copy the file or you can use an access control string.

4.5.10 Example

In the following example, the latest version of all files in the default directory are copied to files with the same names in the directory DISK2:[STAFF_BACKUP] on node CHAOS:

```
$ COPY *.* CHAOS::DISK2:[STAFF_BACKUP]
```

4.5.11 Using Mail to Copy Files

If the file is yours, you can use Mail to send it to a user account on the other node.

When sending files though mail, note the following restrictions:

- When files are copied using the COPY command, the operating system performs data-integrity checking. This check does not occur when sending a file through mail and can cause corrupted files to occur when sending foreign (such as executable) files.
- Use discretion when sending large files. Users on some systems may not be able to receive large files (such as POSTSCRIPT files).

For more information on using Mail to send files, see Chapter 6.

4.5.12 Example

In the following example, the file FEES.DAT is sent to the JONES account on node CHAOS:

```
$ MAIL/SUBJECT="Fee schedule"  FEES.DAT  CHAOS::JONES
```

4.5.13 Using Access Control Strings to Copy Files

To copy files after you have received a protection violation, you can follow the node name in the file specification with an access control string (see Section 4.2.22).

4.5.14 Example

In the following example, the user has an account on node CHAOS with the user name SMITH and the password SPG96PRT. The user is copying all of the files in the default directory to the account on CHAOS.

```
$  COPY *.* CHAOS"SMITH SPG96PRT"::DISK2:[STAFF_BACKUP]
```

In the following example, the user SMITH has WRITE access to the [STAFF_BACKUP]directory.

4.5.15 Renaming Files

Use the RENAME command to give the file a new name and optionally to locate it in a different directory. Note that after being renamed, the original file no longer exists. When you use the RENAME command, the input and output locations must be on the same device.

4.5.16 Example

In the following example, the file FEES.DAT is given the new name RECORDS.DAT and it is moved from the default directory to the [SAVETEXT] directory:

```
$ RENAME FEES.DAT;4 [SAVETEXT]RECORDS.DAT
```

4.6 Displaying the Contents of Files

4.6.1 Using the TYPE Command

To display the contents of a file on your screen, enter the TYPE command and the file name at the DCL prompt. You do not have to specify the version number in the file specification because the system displays the latest version of a file by default.

4.6.2 Example

In the following example, the latest version of the file STAFF_VACATIONS.TXT is displayed:

```
$ TYPE STAFF_VACATIONS.TXT
```

4.6.3 Controlling the Display

To stop the **scrolling** of the text on the screen temporarily, press the Hold Screen key (F1 on VT200 and VT300 series terminals); to resume scrolling, press the Hold Screen key again. To stop the display and return to DCL command level, press Ctrl/Y or Ctrl/O.

If you specify the /PAGE qualifier to the TYPE command, you can view one screen at a time. The system prompts you to press Return when you want to see the next screen.

4.6.4 Using Text Editors to Display Files

By invoking an interactive text editor (for example, EVE or EDT) with the /READ_ONLY qualifier, you can use interactive editing commands to move around in a file and search for specific sequences of characters. The /READ_ONLY qualifier prevents you from creating a modified version of the file when you exit from the interactive editor.

4.6.5 Displaying Files on Remote Nodes

To display the contents of a file on a remote node, include the node name, disk, and directory in the file specification.

4.6.6 Example

In the following example, the file COMPANY_HOLIDAYS.TXT (which is located on remote node CHAOS) is displayed:

```
$ TYPE CHAOS::DISK2:[PUBLIC]COMPANY_HOLIDAYS.TXT
```

4.6.7 Displaying Files with Wildcards

You can use the asterisk (*) wildcard to display all versions of a specific file.

4.6.8 Examples

- In the following example, all versions of the file LOGIN.COM in the directory [JONES] are displayed:

  ```
  $ TYPE [JONES]LOGIN.COM;*
  ```

- In the following example, all versions and all file types of all files that begin with the word STAFF in the directory [JONES] are displayed:

  ```
  $ TYPE [JONES]STAFF*.*;*
  ```

4.6.9 Displaying Multiple Files

If you specify more than one file in the TYPE command line, the system displays the files in the order you specify. If you use wildcard characters, the system displays the files in alphabetical order.

4.7 Deleting Files

4.7.1 Using the DELETE Command

The DELETE command removes files from directories and releases the disk space they occupy for use by other files. When you use the DELETE command, you must specify a version number or the asterisk (*) wildcard character as a version number in each file specification.

4.7.2 Examples

- In the following example, version 17 of the file POUND.LIS is deleted:

```
$ DELETE POUND.LIS;17
```

- In the following example, versions 16 and 17 of the file POUND.LIS are deleted:

```
$ DELETE POUND.LIS;16,;17
```

- In the following example, all versions of the file POUND.LIS are deleted:

```
$ DELETE POUND.LIS;*
```

4.7.3 DELETE Command Qualifiers

When you delete many files with wildcard characters, you might want to confirm each deletion by using the /CONFIRM qualifier. Similarly, you might want to display the names of files as they are deleted. To do this, specify the /LOG qualifier with the DELETE command.

4.7.4 Examples

- In the following example, the deletion of all the files in the subdirectory [JONES.LICENSES.DOG] are confirmed because the /CONFIRM qualifier is specified:

```
$ DELETE/CONFIRM *.*;*
DISK1:[JONES.LICENSES.DOG]FEES.DAT;4, delete? [N]: Y
DISK1:[JONES.LICENSES.DOG]FEMALE.LIS;6, delete? [N]: Y
DISK1:[JONES.LICENSES.DOG]MALE.LIS;3, delete? [N]: N
DISK1:[JONES.LICENSES.DOG]POUND.LIS;17, delete? [N]: Y
```

- In the following example, the system displays the names of the files after they are deleted because the /LOG qualifier is specified:

```
$ DELETE/LOG *.LIS;*
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]FEMALE.LIS;6 deleted (35 blocks)
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]MALE.LIS;3 deleted (5 blocks)
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]POUND.LIS;17 deleted (9 blocks)
```

4.7.5 Using the PURGE Command

The PURGE command deletes all except the latest version of the specified file (or all files) in the default directory or any other specified directory. Purging old versions of files after updating them enables you to retain more free space on your disk.

4.7.6 Example

In the following example, all except the latest two versions of each file in the default directory are purged:

```
$ PURGE/KEEP=2
```

4.8 Protecting Files from Other Users

4.8.1 Access Control Lists (ACLs)

To prevent other users from accessing your files, you can change the protection or modify the access control list (ACL) of your files. To change the protection or modify the ACL of a file, you must own the file, have control access to the file, or have GRPPRV, SYSPRV, BYPASS, or READALL privilege.

4.8.2 Types of Protection

There are two types of file protection: default and explicit. When a file is created, it usually has the same protections as its parent directory; this is the default protection. If you create a file using the CREATE/PROTECTION command or if you change the protection on an existing file by issuing the SET SECURITY /PROTECTION command, you are using explicit file protection.

Note that to protect a file completely, you must apply the same or greater protection to the directory in which the file resides.

4.8.3 For Additional Information

For additional security information, refer to:

- Chapter 5 for information on directory protection
- Chapter 19 for complete information on changing file protections

4.9 Printing Files

4.9.1 Using the PRINT Command

To print a file or files, use the PRINT command. The PRINT command places your print job (all the files to be printed) in a list of jobs to be printed called a **print queue**. The file types of the files named in the PRINT command default to .LIS or the last explicitly named file type. The system displays the job name, the queue name, the job number, and status of the job.

By default, the job name is the name of the first (or only) file specification in the PRINT command. After a job is submitted to a queue, you reference it using the job number. After the job is queued, it will be printed when no other jobs precede it in the queue and when the printer is physically ready to print.

4.9.2 Example

In the following example, a print job containing three files is placed in the default print queue, SYS$PRINT:

```
$ PRINT POUND,MALE,FEES.DAT
Job POUND (queue SYS$PRINT, entry 202) started on SYS$PRINT
```

Because the default file type for the PRINT command is .LIS, the files POUND.LIS, MALE.LIS, and FEES.DAT are queued. The job name is POUND, the queue name is SYS$PRINT, and the job number is 202.

4.9.3 Print Job Priority

A print queue can execute only one job at a time. Print jobs are scheduled for printing according to their scheduling **priority**, and the job with the highest priority is printed first. If more than one job exists with the same priority, the smallest job is usually printed first. Jobs of equal size having the same priority are selected for printing according to their submission time. Priority may also be determined by the system manger or by entering the /PRIORITY qualifier to the PRINT command. For more information on scheduling priorities, see the *OpenVMS System Manager's Manual*.

4.9.4 Displaying Queue Information

The default print queue, SYS$PRINT, is usually started as part of the site-specific system startup procedure. The following table shows commands you can use to display information about queues:

To display...	Enter this command...
The queues at your site	SHOW QUEUE
The status of your print jobs	SHOW ENTRY
Jobs queued by other users	SHOW ENTRY/USERNAME= *username*
Information about a specific job or jobs	SHOW ENTRY *job-name* SHOW ENTRY *entry-number*

4.9.5 Example

In the following example, the SHOW ENTRY command is used to display information about a print job that has been queued:

```
$ SHOW ENTRY

  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
    202  POUND           JONES            38  Pending
         On stopped printer queue SYS$PRINT)
```

4.9.6 Print Forms

A **print form** serves the following functions:

- Determines certain page formatting attributes (such as margins and page length)
- Determines whether a job is eligible to print depending on the paper stock specified in the form

If your printing needs are limited, you do not need to use special forms because Digital supplies a systemwide default form (named DEFAULT) for all queues. System managers can also create print forms. If you need to format output or if certain print jobs require special paper, contact your system manager.

4.9.7 Stopping a Print Job

To stop a print job and delete it from the print queue, enter the entry number parameter to the DELETE/ENTRY command.

4.9.8 Example

In the following example, entry 202 is deleted:

```
$ DELETE/ENTRY=202
```

4.9.9 Printing Files on Other Nodes

To print a file on another system, copy that file to the remote node and specify the /REMOTE qualifier to the PRINT command.

4.9.10 Example

In the following example, the file COMPANY_HOLIDAYS.TXT is copied from the local node to the remote node CHAOS and the file is queued for printing to the default system print queue (SYS$PRINT) on node CHAOS:

```
$ COPY COMPANY_HOLIDAYS.TXT CHAOS"JONES PANDEMONIUM"::DISK2:[JONES]*
$ PRINT/REMOTE CHAOS::DISK2:[JONES]COMPANY_HOLIDAYS.TXT
```

An access control string indicates that the user JONES is authorized to copy files to the directory [JONES] on node CHAOS. The asterisk (*) wildcard at the end of the file specification instructs the system to duplicate the file name COMPANY_ HOLIDAYS.TXT when that file is copied to the remote node.

4.9.11 Using the /REMOTE Qualifier

Not all qualifiers to the PRINT command are compatible with the /REMOTE qualifier. For example, you cannot queue a job to a specific print queue; all jobs are queued to the default system print queue (SYS$PRINT). See the description of the /REMOTE qualifier to the DCL command PRINT in the *OpenVMS DCL Dictionary* for a list of PRINT command qualifiers compatible with /REMOTE.

4.9.12 Print Command Qualifiers

Print jobs can be controlled in various ways by using qualifiers to the PRINT command. For example, you can specify the number of copies printed or you can request that the system notify you when your print job is complete.

In addition to the qualifiers described in this manual, if you are running DECprint Supervisor software on your system, you can use the /PARAMETER qualifier to print landscape, two-sided, or many other ways. Contact your system manager for a list of print options that are available on your system.

4.9.13 Summary of Commands That Control Print Jobs

The following table lists a summary of PRINT command qualifiers. For complete information on the PRINT command, refer to the *OpenVMS DCL Dictionary* or online help.

Print Operations	Print Job Commands and Qualifiers
Number of copies	
By job	PRINT/JOB_COUNT=n[1]
By file	PRINT/COPIES=n[1]
Specified file only	file-spec/COPIES=n[1]
Number of pages	PRINT/PAGES=[1]
Print features	
Flag pages	PRINT/FLAG=[1]
Type of forms (paper)	PRINT/FORM=[1]
Special features	PRINT/CHARACTERISTICS=[1]
Double-spacing	PRINT/SPACE[1]
Page heading	PRINT/HEADER[1]
Notification of job execution	PRINT/NOTIFY
Delay execution of a job	
For a specified time	PRINT/AFTER
Indefinitely	PRINT/HOLD
Release a delayed job	SET QUEUE/ENTRY/RELEASE
Display your print jobs	SHOW ENTRY
Stop a print job	
Delete job	DELETE/ENTRY=job-number
Stop current job and begin printing the next job in the queue	STOP/ABORT
Stop current job and requeue it for printing	STOP/REQUEUE
Keep a job in a queue after it has completed	PRINT/RETAIN

[1]Parallel qualifiers for the SET QUEUE/ENTRY command allow you to specify these operations for print jobs that are already queued but not yet printing.

5

Directories: Organizing and Managing Files

5.1 Overview

Directories are files that store the names of files. This chapter describes how to use directories to organize and manage files. This chapter includes information on:

- Directory structures
- Understanding directories
- Defaults
- Protecting directories from other users
- Using wildcards to search the directory structure
- Working with directories in UIC format

5.1.1 Note: Examples Used in This Chapter

Throughout this chapter, examples that specify a node name do not always include an access control string. This is because proxy accounts enable users to perform operations on the remote systems in these examples.

5.1.2 References

For more information on the DCL commands described in this chapter, refer to DCL Help or see the *OpenVMS DCL Dictionary*.

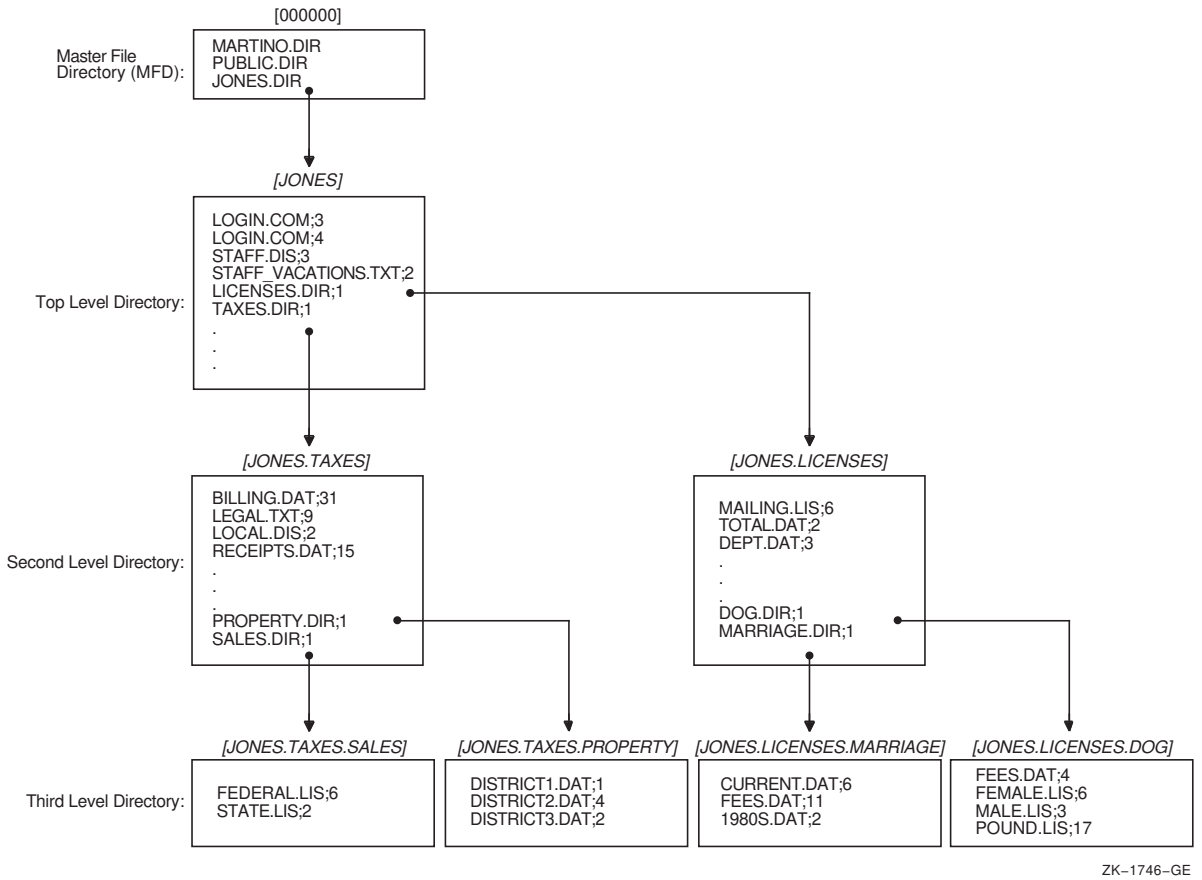
5.2 Directory Structures

5.2.1 Example of a Directory Structure

Figure 5–1 shows a sample directory hierarchy. At the top of the structure is the master file directory (MFD). Its directory name is [000000]. The MFD shown contains entries for user file directories including MARTINO.DIR, PUBLIC.DIR, and JONES.DIR. The top-level directory [JONES] is a user file directory named JONES.DIR;1 in [000000].

The sample directory structure in Figure 5–1 is the basis for many of the examples in this chapter.

Figure 5–1 Directory Structure



Note the following about this directory structure:

1. Assume that you are user JONES. When you log in, the system places you in [JONES], your default directory.
2. [JONES] contains the following four nondirectory files:
   - LOGIN.COM;3
   - LOGIN.COM;4
   - STAFF.DIS;3
   - STAFF_VACATIONS.TXT;2
3. [JONES] also contains the following two directory files:
   - LICENSES.DIR;1
   - TAXES.DIR;1
4. The directory file LICENSES.DIR;1 points to the [JONES.LICENSES] subdirectory.
5. TAXES.DIR;1 points to the [JONES.TAXES] subdirectory.
6. The [JONES.LICENSES] subdirectory contains three nondirectory files and two directory files.

7. The directory file DOG.DIR;1 points to the [JONES.LICENSES.DOG] subdirectory.
8. MARRIAGE.DIR points to the [JONES.LICENSES.MARRIAGE] subdirectory.

5.3 Understanding Directories

5.3.1 Directory Specifications

Use a directory specification to refer to a directory. A directory specification consists of a top-level directory name that can be followed by a maximum of seven subdirectory names. Subdirectory names are always preceded by a period (.).

5.3.2 Directory Specification Format

A directory specification has the following format:

[directory.subdirectory]

To add one or more levels of subdirectories, add a period and another subdirectory name for each subdirectory (up to seven levels): (Subdirectories are specified by concatenating the subdirectory name to the name of the directory one level above it.)

[directory.subdirectory.subdirectory]

A directory or subdirectory name can contain up to 39 alphanumeric characters. Any characters valid for file names are also valid for directory names. Enclose the directory name in either square brackets ([ ]) or angle brackets (< >).

5.3.3 Creating Directories

To create a directory, enter the CREATE/DIRECTORY command. If you want to create a subdirectory under your current directory, you do not have to specify the current directory name; you can enter the subdirectory name preceded by a period.

5.3.4 Examples

- In the following example, the directory [JONES.LICENSES] is created:

```
$ CREATE/DIRECTORY [JONES.LICENSES]
```

- In the following example, the current default directory is [JONES], and the subdirectory [JONES.LICENSES] is created:

```
$ CREATE/DIRECTORY [.LICENSES]
```

5.3.5 Displaying Directories

To display the names of files in a directory, enter DIRECTORY at the DCL prompt. To list the files in a subdirectory, enter the DIRECTORY command and the subdirectory name preceded by a period.

When you include certain command qualifiers along with the DIRECTORY command, you can retrieve information in addition to the names of the files. For more information on DIRECTORY command qualifiers, refer to the *OpenVMS DCL Dictionary* or online help.

5.3.6 Examples

- In the following example, the files in the directory [JONES] are listed. The example shows that [JONES] contains two subdirectories, [JONES.LICENSES] and [JONES.TAXES], four nondirectory files, STAFF.DIS, STAFF_VACATIONS.TXT, and two versions of LOGIN.COM:

```
$ DIRECTORY

Directory DISK1:[JONES]

LICENSES.DIR;1
LOGIN.COM;3
LOGIN.COM;4
STAFF.DIS;3
STAFF_VACATIONS.TXT;2
TAXES.DIR;1

Total of 6 files.
```

- In the following example, the default directory remains [JONES] and the contents of the subdirectory [JONES.LICENSES] are displayed:

```
$ DIRECTORY [.LICENSES]

Directory DISK1:[JONES.LICENSES]

DEPT.DAT;3
DOG.DIR;1
MAILING.LIS;6
MARRIAGE.DIR;1
TOTAL.DAT;2

Total of 5 files.
```

5.3.7 Deleting Directories

To delete a directory, use the following procedure:

Step	Task
1	Make sure that the directory contains no files. To find out if the directory contains files, enter the DIRECTORY command. When there are no files in the directory, the system displays the following message: `%DIRECT-W-NOFILES, no files found`

Step	Task
2	If the directory contains files, copy them to another directory to save them or delete them if you do not want to save them. If the directory contains subdirectories, examine those subdirectories, copy or delete their files, and delete the subdirectories.
3	Move to the directory one level above the directory you want to delete. Remember that subdirectories exist as files in directories. When you delete a directory, you delete the file that points to that directory.
4	Change the file protection of a directory to allow delete access to the file. Directory files in master file directories require SYSPRV privilege to delete. (See Chapter 4 for more information about file protection.)
5	Delete the directory file using the DELETE command.

5.3.8 Example

The following example shows how to delete the subdirectory [JONES.LICENSES]:

```
$ SET DEFAULT [JONES.LICENSES]
$ DIRECTORY
%DIRECT-W-NOFILES, no files found
$ SET DEFAULT [JONES]
$ SET SECURITY/PROTECTION=OWNER:D LICENSES.DIR
$ DELETE LICENSES.DIR;1
```

5.4 Defaults

5.4.1 Changing Your Default Directory

To change your default directory, use the SET DEFAULT command. The new default remains in effect until you enter another SET DEFAULT command or log out. To set default to a subdirectory, append the subdirectory name to the name of the directory one level above it.

5.4.2 Examples

- In the following example, default is set to the directory [JONES] and then the file [JONES]STAFF_VACATIONS.TXT is displayed:

  ```
  $ SET DEFAULT [JONES]
  $ TYPE STAFF_VACATIONS.TXT
  ```

- In the following example, the file BILLING.DAT, which is located in the subdirectory [JONES.TAXES], is displayed:

  ```
  $ SET DEFAULT [JONES.TAXES]
  $ TYPE BILLING.DAT
  ```

5.4.3 Setting Default to Nonexistant Directories

Note that the operating system allows you to set default to a nonexistent disk or directory. If you have set default to a nonexistent directory, when you try to manipulate a file, the system displays a message stating that the directory does not exist. If you find yourself in a nonexistent disk or directory and cannot carry out a desired operation, set default to an existing disk or directory.

5.4.4 SHOW DEFAULT Command

To display your current default directory, enter the command SHOW DEFAULT, as shown in the following example:

```
$ SHOW DEFAULT
  DISK1:[JONES.TAXES]
$ SET DEFAULT [PUBLIC]
$ SHOW DEFAULT
  DISK1:[PUBLIC]
```

5.4.5 Setting Default Devices

You can use the SET DEFAULT command to change the default device. The default remains in effect until you enter another SET DEFAULT command or log out. You can also specify the device to which you want to set default without including the directory in the command.

5.4.6 Examples

- The following example shows how to change the default device:

```
$ SHOW DEFAULT
  DISK1:[JONES]
$ SET DEFAULT DISK2:[GROUP]
$ SHOW DEFAULT
  DISK2:[GROUP]
```

- In the following example, the directory [JONES] is assumed and exists on DISK1 and DISK2:

```
$ SHOW DEFAULT
  DISK1:[JONES]
$ SET DEFAULT DISK2:
$ SHOW DEFAULT
  DISK2:[JONES]
```

5.4.7 Using Temporary Defaults

If you enter a list of files and do not give a complete file specification for each file in the list, the system applies temporary defaults for node names, device names, and directory names. To substitute your current default directory for a temporary default, use empty square brackets. If you include a node name in a file that appears in a list, you can override the temporary default by using a double colon.

5.4.8 Examples

- In the following example, A.LIS and B.LIS are copied from the [STATS] directory to the [RESULTS] directory:

  ```
  $  COPY [STATS]A.LIS,B.LIS  [RESULTS]
  ```

  Note that the system uses the preceding file specification in the list, [STATS]A.LIS, to determine that the temporary default directory for file B.LIS is [STATS] as well.

- In the following example, a temporary default device and two different directories are used:

  ```
  $  COPY BASE:[STATS]A.LIS,[TIME]B.LIS,C.LIS  [RESULTS]
  ```

  All three files (A.LIS, B.LIS, and C.LIS) are copied from the BASE device. The A.LIS file is copied from the [STATS] directory. The other two files are copied from the [TIME] directory.

- In the following example, the current default directory is [BETA]. This command copies [ALPHA]TEST.DAT and [BETA]FINAL.DAT to the [RESULTS] directory:

  ```
  $  COPY [ALPHA]TEST.DAT,[]FINAL.DAT  [RESULTS]
  ```

5.5 Protecting Directories from Other Users

5.5.1 Reasons to Protect Directories

You cannot completely protect a file without applying at least the same protection to the directory in which the file resides. For example, if you deny a user all access to a file but allow that user read access to the file's directory, the user cannot access the contents of the file but can see that it exists. Conversely, a user allowed access to a file and denied access to the file's directory (or one of the parent directories) cannot see that the file exists.

5.5.2 Private Files

To protect private files, directory protection alone is not adequate. You must also protect each file within the directory.

5.5.3 Default Directory Protection

By default, top-level directories receive UIC-based protection (S:RWE,O:RWE,G:RE,W:E) and no ACL. Subdirectories receive UIC-based protection from the parent directory. For more information on protection codes, see Section 19.4.

5.5.4 UIC-Based Protection

To specify UIC-based protection explicitly when creating a directory, use the /PROTECTION qualifier with the CREATE /DIRECTORY command. You cannot specify an ACL for the directory until the directory is created. To change the UIC-based protection of an existing directory, apply the SET SECURITY /PROTECTION command to the directory file.

5.5.5 Limiting Directory Access

You can limit but not prohibit directory access by specifying execute access but not read access. Execute access on a directory permits you to examine and read files that you know are contained in the directory; that means you can examine a file if you already know what the file specification is, but you cannot display a list of the files in the directory. For additional security information see *OpenVMS Guide to System Security*.

5.6 Using Wildcards to Search the Directory Structure

5.6.1 Overview

From any point in a directory structure, you can refer to another directory or subdirectory in the structure. Do this by specifically naming the directory or subdirectory you want or by using the ellipsis (. . .) and hyphen (-) wildcard characters. For additional information about wildcards, see Section 4.3.

5.6.2 Ellipsis Wildcard Character

Use the ellipsis (. . .) wildcard character to search down into the directory hierarchy. To search the current directory and all the subdirectories below it, use the ellipsis by itself as shown:

```
$ DIRECTORY [...]
```

If you begin the directory specification with an ellipsis, the search begins from your current directory. However, if you begin the directory specification with a period, only the subdirectory that is one level lower than the current directory is searched.

To search all top-level directories and their subdirectories from wherever you are in the directory structure, use an asterisk (*) followed by an ellipsis (. . .).

5.6.3 Examples

- In the following example, assuming the current directory is [JONES], the latest versions of all files named FEES.DAT in [JONES] and all subdirectories under [JONES] will be displayed:

  ```
  $ TYPE [JONES...]FEES.DAT
  ```

- In the following example, assuming the current default directory is [JONES], all subdirectories that end in .SALES are searched, and the latest versions of the file FEDERAL.LIS are displayed:

  ```
  $ TYPE [...SALES]FEDERAL.LIS
  ```

- In the following example, the latest versions of all files named DEPT.DAT in [JONES] and all subdirectories under [JONES] are displayed:

  ```
  $ TYPE [...]DEPT.DAT
  ```

- In the following example, assuming the current directory is [JONES], the [.LICENSES] subdirectory will be searched for the file MAILING.LIS, but [JONES.LICENSES.MARRIAGE] will not:

```
$ TYPE [.LICENSES]MAILING.LIS
```

- In the following example, assuming the current directory is [JONES], the latest versions of all files named DEPT.DAT in the [.LICENSES] subdirectory under [JONES]and all subdirectories under the [.LICENSES] subdirectory are displayed:

```
$ TYPE [...LICENSES...]DEPT.DAT
```

- In the following example, as many as eight levels of directory names (the top-level directory and seven subdirectories) are searched (if they exist). Note that the command shown requires READALL privilege.

```
$ DIRECTORY [*...]
```

5.6.4 Hyphen Wildcard Character

Use the hyphen (-) wildcard character to move up through the directory structure. Each hyphen refers to the directory one level up from the current one. You can follow the hyphens with directory and subdirectory names to move down the directory structure on another path.

You can specify more than one hyphen. If you enter so many hyphens that you point above the top-level directory, the system displays an error message.

5.6.5 Examples

- In the following example, the current directory is [JONES.LICENSES]. The following command displays the latest version of STAFF.DIS in [JONES]:

```
$ TYPE [-]STAFF.DIS
```

- In the following example, the current directory is [JONES.LICENSES]. The command shown displays the latest version of BILLING.DAT in [JONES.TAXES]:

```
$ TYPE [-.TAXES]BILLING.DAT
```

- In the following example, the command shown moves you up two levels in the directory hierarchy:

```
$ SET DEFAULT [--]
```

5.7 Working with Directories in UIC Format

5.7.1 Overview

Although this chapter focuses on how to use named directories, you can also specify directory names in UIC format. In UIC format, a 2-part octal number forms a user identification code (UIC) that refers to a user file directory (UFD). Almost every DCL command that accepts a file specification can recognize directory names in UIC format. In general, you do not need to use this format unless you are working with a real-time Resource Sharing Executive (RSX) operating system.

5.7.2 UIC Directory Format and Rules

A UIC directory specification has the following format:

[group,member]

For example, [122,1] is a UIC directory specification representing member 1 in group 122. Directory names in UIC format generally, but not necessarily, correspond to the UIC of the owner of the directory.

When you refer to a UIC directory, observe the following rules:

- Use an octal number in the range of 1 to 37776 to specify the group.
- Use an octal number in the range of 0 to 177776 to specify the member.
- Do not use the hyphen (-) or ellipsis (. . .) wildcard as part of the specification.

5.7.3 Using Wildcards with UIC Directories

It is also possible to use the asterisk (*) wildcard to specify a UIC directory. For example, [*,6] indicates all directories with any group number and a member number of 6. The search is limited to directories in UIC format. The directory specification [*,*] locates all directories in UIC format. To locate all named directories as well as all directories in UIC format, use [*].

5.7.4 Translating to Named from UIC Format

Note that you can translate a directory name in UIC format to named format. If necessary, add zeros to the left of the group and member numbers to create a 6-character name.

You cannot combine UIC format and named format. If you have a directory with a name in UIC format and you want to specify one of its subdirectories, translate the UIC format to named format.

5.7.5 Examples

- The named equivalent of the UIC directory specification [122,1] is as follows:

  ```
  [122001]
  ```

- To refer to the subdirectory [122,1]SUB.DIR, use the named directory [122001.SUB].

6

Mail: Communicating with Other Users

6.1 Overview

The OpenVMS Mail utility (MAIL) lets you send messages to other users on your system or on any other computer that is connected to your system with the DECnet for OpenVMS network. This chapter describes:

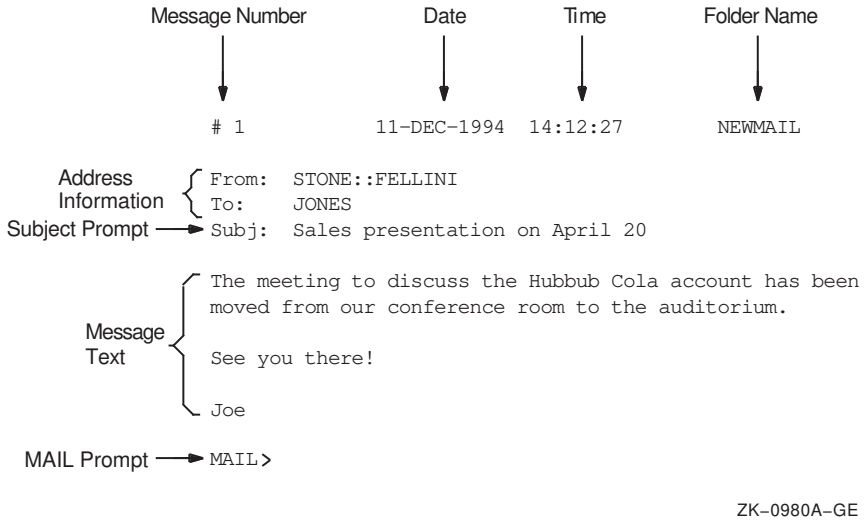
- Invoking and exiting mail
- Reading messages
- Sending messages
- Sending mail over networks
- Sending messages to multiple users
- Manipulating files in mail
- Other ways to send messages
- Organizing messages
- Deleting messages
- Printing mail messages
- Protecting mail files
- Using text editors in Mail
- Customizing your Mail environment
- Summary of Mail commands

6.1.1 References

- For additional information on commands and qualifiers, enter the HELP command at the MAIL> prompt.
- For information on controlling the use of Mail through user accounts, see the *OpenVMS System Manager's Manual*.
- For additional information on MAIL command qualifiers, refer enter the HELP MAIL command at the DCL prompt.

6.1.2 Figure: Sample Mail Message

The following figure shows a sample mail message and its components:



6.2 Invoking and Exiting Mail

6.2.1 Invoking Mail

To invoke the Mail utility, enter the DCL command MAIL, as follows:

```
$ MAIL   Return
MAIL>
```

Once you are in the Mail utility, you can enter Mail commands by entering a command at the MAIL> prompt and pressing either the Return or Enter key.

6.2.2 Using Mail

By entering Mail commands at the MAIL> prompt, you can do the following:

- Read a mail message
- Send a mail message
- Reply to a mail message
- Forward a mail message
- Organize mail messages into files and folders
- Delete a mail message
- Print a mail message

6.2.3 Exiting from Mail

To exit from Mail, enter the EXIT command at the MAIL> prompt, as follows:

```
MAIL> EXIT   Return
$
```

You can also exit from Mail by pressing Ctrl/Z or by using the QUIT command.

6.3 Reading Messages

6.3.1 Overview

Mail stores the messages you receive in mail files, which have the default file type .MAI. In this file, by default, Mail provides two **folders** that store old and new messages. New messages are automatically placed in a folder called NEWMAIL; old messages are placed in a folder called MAIL. After you read a new message, the message automatically moves from the NEWMAIL folder to the MAIL folder, unless you enter the FILE, MOVE, or DELETE command. Mail deletes the NEWMAIL folder after you have read all new mail messages and either select another folder or exit from Mail.

6.3.2 New Mail Notification

When you are logged in to your account and receive a mail message, Mail notifies you. For example, notification of a message sent by user FELLINI is displayed as follows:

```
New mail on node DOODAH from STONE::FELLINI     (10:02:23)
```

6.3.3 Reading New Mail

To read a new message, invoke Mail and press the Return key at the MAIL> prompt, as follows:

```
$  MAIL
You have 1 new message.
MAIL>  Return
```

6.3.4 Reading More Than One Message

If you have more than one new message, press Return at the MAIL> prompt to read the other messages. When you have read all your new messages, Mail issues the message ‘‘%MAIL-E-NOMOREMSG, no more messages’’ and continues to prompt for commands until you exit Mail.

6.3.5 Reading New Mail While in Mail

If you receive a mail message while you are in Mail, enter the READ/NEW command to read the new message.

6.3.6 Reading Old Messages

To reread old mail messages in your default Mail folder, use the following procedure:

Step	Task
1	Enter the SELECT command at the MAIL> prompt. For example: `MAIL> SELECT MAIL` Return Mail places you in the folder named MAIL.
2	To read the first message in your default MAIL folder, press Return at the MAIL> prompt or enter the READ command. Mail displays the first message (1) in your default mail file.
3	To display the next message, press Return. If the message is too long to display on one screen, press Return to display the next part of the message. To skip the remainder of a message and display the next message, enter the NEXT command.

6.3.7 Reading Specific Old Mail

To read a particular message in your default MAIL folder, use the following procedure:

Step	Task
1	Enter the DIRECTORY command at the MAIL> prompt. To select a subset of messages from the list, use the DIRECTORY command qualifiers /FROM or /SUBJECT.
2	Enter the number of the message that you want to read at the MAIL> prompt. Mail displays the message that you selected.

6.3.8 Example

In the following example, the DIRECTORY command is used to display old messages and then the message labeled 2 is selected for reading:

```
MAIL> DIRECTORY Return
                                                          MAIL
# From              Date              Subject
1 STONE::FELLINI    11-DEC-1995       Sales presentation on May 11
2 DOODAH::JONES     11-DEC-1995       Status

MAIL> 2 Return
```

6.3.9 Searching for Messages

If you have many messages, you can locate a particular message by using the SEARCH command to find a string in one or more of the messages. To search for a string, specify that string as a parameter to the SEARCH command.

Each time you specify a new string, the SEARCH command starts the search at message number 1. To continue searching the folder for messages that contain the specified string, use the SEARCH command without specifying a parameter. To search for the same string in a different folder, enter the SELECT or SET FOLDER folder-name command and continue using the SEARCH command without specifying a parameter.

6.3.10 Example

In the following example, messages in the current folder are searched for the first messages that contains the string *appointment*:

```
MAIL> SEARCH "appointment"  Return
```

6.4 Sending Messages

6.4.1 How to Send Mail

To send a mail message to any user on your system, do the following:

Step	Task
1	Enter SEND at the MAIL> prompt. Mail prompts you for the name of the user to receive the message.
2	Enter the name of the user receiving the message and press Return. Mail prompts you for the subject of the message.
3	Enter the subject of the message and press Return. Entering this information is optional. Mail prompts you for the text of the message.
4	Enter the text of a message, or just press Return. Entering this information is optional.
5	Press Ctrl/Z to send the message. If you decide not to send the message, press Ctrl/C, which cancels the send operation without exiting from Mail.

6.4.2 Example

In the following example, a message is sent to a user named THOMPSON:

```
MAIL> SEND  Return
To: THOMPSON  Return
Subj: Meeting on April 20  Return
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:
I have some new ideas about the Hubbub Cola account.  Return
Let me know  when you are available to talk about them.  Return
 Return

--Jeff
```

6.5 Sending Mail Over Networks

6.5.1 Specifying Node Names

If your computer system is part of a network, you can send mail to any other user on the network. If you are sending mail to someone on a different node, enter the user's node name and user name at the To: prompt. If the user name contains special characters or spaces, you must enclose the user name in quotation marks (""). Use the following format:

nodename::username

Mail displays a message if the network connection to the remote node is not available. Wait a while, and try to send the message later.

For additional information on specifying node names, refer to Section 4.2.9.

6.5.2 Example

In the following example, a message is sent to user HIGGINS on node CHEETA:

```
MAIL> SEND Return
To: CHEETA::HIGGINS Return
```

6.5.3 Using Logical Node Names

You can use a logical name to represent a user's name and node; then you can use the logical name to send mail. Note that Mail ignores any access control information in the node name or logical name.

6.5.4 Example

In the following example, HENRY is used in place of CHEETA::HIGGINS. First, the logical name (HENRY) is defined, then it is used in place of the username and node:

```
$ DEFINE HENRY CHEETA::HIGGINS
$ MAIL Return
MAIL> SEND Return
To: HENRY Return
```

6.6 Sending Messages to Multiple Users

6.6.1 Using Individual Names

You can send mail to several users at the same time in one of two ways: using individual user names at the To: prompt or using a distribution list. To send the same message to several users on the same node by using their user names, enter the user names at the To: prompt and separate them with commas or spaces.

6.6.2 Example

In the following example, a message is sent to Thompson, Jones, and Barney:

```
MAIL> SEND Return
To:      THOMPSON,JONES,BARNEY Return
Subj:    Meeting on January 9 Return
```

6.6.3 Distribution Lists

A distribution list is a file that contains a list of users and their node names. You must use a text editor to create distribution lists. Distribution lists are not created within the Mail utility.

Your open file quota (a limit associated with your account) determines the number of different nodes to which you can send mail (at one time) and the depth to which you can nest distribution lists. If you exceed the quota, Mail displays an error message. Ask your system manager to increase your quota or send mail in batches to fewer nodes at one time.

6.6.4 Default File Type for Distribution Lists

By default, the system looks for a distribution list file with the file type .DIS. If the file containing your distribution list has a different file type, specify the file name and file type at the To: prompt. If you invoke Mail while in one directory and the file containing the distribution list is in another, enter the distribution list's full directory name at the To: prompt.

6.6.5 How to Create Distribution Lists

To create a distribution list, use the following procedure:

Step	Task
1	Use a text editor to create a distribution list file with the file type .DIS.
2	Type one user name per line in the file.
3	To include the names of other distribution lists in the file (to ''nest'' the lists), specify an at sign (@) followed by the name of the distribution list.
4	To include comments in the file, enter an exclamation point (!) before the comment.

6.6.6 Example

The following example shows a distribution list file:

```
! ALLBUDGET.DIS
!
! Budget Committee Members
@BUDGET         ! listed in BUDGET.DIS.
! Staff
  Thompson
  BRUTUS::JONES
  PORTIA::BARNEY
```

In the following example, if the file BUDGET.DIS is not in the same directory as the new distribution list file you are creating (ALLBUDGET.DIS), include the file specification for BUDGET.DIS in the new distribution file. Depending on where you create ALLBUDGET.DIS, you might have to specify the device and directory in which BUDGET.DIS is located. (See Chapter 4 for more information about file specifications.)

6.6.7 Sending Messages to Distribution Lists

To send mail to several users by using a distribution list, use the following procedure:

Step	Task
1	Invoke Mail.
2	Enter SEND at the MAIL> prompt and press Return.
3	Enter an at sign (@) and the file name of the distribution list at the To: prompt. Press Return.
4	Enter the subject of the message at the Subj: prompt and press Return.
5	Enter the text of the message at the text prompt.

6.6.8 Example

In the following example, a message is sent to the distribution list ALLBUDGET.DIS:

```
MAIL> SEND [Return]
To: @ALLBUDGET [Return]
Subj: Tomorrow's Meeting [Return]
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:

The meeting about the Hubbub Cola account is tomorrow at 2:00.  [Return]

--Jeff  [Return]
```

6.6.9 Sending Mail to Distribution Lists from DCL

You can also send a file to a distribution list from DCL level. If you omit the file type .DIS, place quotation marks ("") around the at sign (@) and file name to identify the file as a distribution list. To include a subject, use the /SUBJECT qualifier with the MAIL command.

6.6.10 Examples

- The following example sends the file MEETING.TXT to the user THOMAS and the distribution list FRIENDS.DIS:

  ```
  $ MAIL/SUBJECT="update" MEETING THOMAS,"@FRIENDS.DIS" [Return]
  ```

- The following example sends the file NOTICE.TXT to the distribution list WRITERS.DIS. Here, the /SUBJECT qualifier is not included so the message is sent without a subject notation.

  ```
  $ MAIL NOTICE "@WRITERS" [Return]
  ```

6.7 Manipulating Files in Mail

6.7.1 How to Send Files

You can send a file to other users from within Mail or from DCL level. Use the following procedure to send a file from within Mail:

Step	Task
1	At the MAIL> prompt, enter SEND and the name of the file you want to send.
2	At the To: prompt, enter the user name of the person you want to receive the file.
3	At the Subj: prompt, enter the subject of the file.
4	Press Return to send the file. To cancel the send operation, press Ctrl/C or Ctrl/Y. Ctrl/C keeps you within Mail; Ctrl/Y returns you to DCL level.

6.7.2 Example

In the following example, the file MEMO.TXT is sent to user EDGELL:

```
MAIL> SEND MEMO.TXT [Return]
To: EDGELL [Return]
Subj: Another memo [Return]
```

6.7.3 Sending DDIF Files

If the file is a compound document structured according to the DIGITAL Document Interchange Format (DDIF) specification, Mail preserves the OpenVMS RMS file tags and DDIF semantics, for OpenVMS AXP Version 1.0 or VAX VMS Version 5.2-2 or later systems only. If you try to send mail messages containing DDIF files to operating systems other than OpenVMS or to OpenVMS systems earlier than OpenVMS AXP Version 1.0 or VAX VMS Version 5.2-2, Mail returns an error message.

6.7.4 Sending Files from DCL

When you send a file from DCL level, Mail is invoked but you do not enter an interactive session, nor do you see the MAIL> prompt. When the file is sent, you return to DCL level automatically. After you have typed the MAIL command with the appropriate qualifiers, press Return to send the file or press Ctrl/C to cancel the send operation.

Note the following as well:

- No wildcard characters are allowed in the file specification. If you omit the file type, the default file type is .TXT
- If you specify SYS$INPUT as the file specification, you are prompted for the text of the message (while still remaining at the DCL level). For more information on using SYS$INPUT, see Chapter 13.

- When you are sending a file from DCL level, the argument to the optional /SUBJECT qualifier must be enclosed in quotation marks if it contains any spaces or nonalphanumeric characters

6.7.5 Examples

- In the following example, the file MEMO.TXT is sent to user EDGELL on node CHEETA from the DCL level:

```
$ MAIL/SUBJECT="Another memo" MEMO.TXT CHEETA::EDGELL Return
```

- In the following example, the user is prompted to input the text of the message because the file name specified is SYS$INPUT:

```
$ MAIL SYS$INPUT: Return
To: ARMSTRONG Return
Enter your message below.  Press CTRL/Z when complete, or CTRL/C to quit:
The text of the message is here.
Ctrl/Z
$
```

6.7.6 Creating Files from Messages

To create a text file from a message, enter the EXTRACT command and the file name at the MAIL> prompt while you are reading the message. When you exit from Mail, the file is listed in your current directory (unless you specify another directory). If the file is a DDIF file, Mail preserves the OpenVMS RMS file tags and DDIF semantics (VMS Version 5.2-2 or later).

The mail header is composed of the From:, To:, and Subj: lines. To create a file that does not include header information, specify the /NOHEADER qualifier to the EXTRACT command. If the message has more than one header (for example, a forwarded message), only the topmost header is deleted.

Use the /APPEND qualifier to the EXTRACT command to copy a message to the end of an existing file. Use the /ALL qualifier to copy all the files in the current folder to an existing file.

6.7.7 Examples

- In the following example, a file named DEC_MEETINGS.TXT is created from the mail message shown:

```
#1                 01-DEC-1995  14:12:27        NEWMAIL

From:  STONE::FELLINI
To:    Thompson
Subj:  Dates for December sales meetings

Sales meetings in December will be held on the following dates:
      Wednesday Dec. 8, 1995
      Tuesday   Dec. 14, 1995
      Monday    Dec. 20, 1995
      Thursday  Dec. 30, 1995
MAIL> EXTRACT DEC_MEETINGS.TXT Return
%MAIL-I-CREATED, DISK:[THOMPSON]DEC_MEETINGS.TXT
```

- The following example shows how to create a file named JANUARY_MEETINGS.TXT containing the text of message number 3:

```
MAIL> READ 3 Return
.
.
.
MAIL> EXTRACT/NOHEADER JANUARY_MEETINGS.TXT  Return
%MAIL-I-CREATED, DISK1:[JONES]JANUARY_MEETINGS.TXT;1 created
MAIL>
```

6.7.8 Appending Files to Messages

To append a small file to the end of a mail message automatically, use the SET SIGNATURE_FILE command. The file you specify is automatically (by default) appended to every mail message you send using the ANSWER, FORWARD, MAIL, REPLY, or SEND command. An example of a signature file is a text file that is formatted as business card, containing the user's company name, address, telephone, and Internet address.

If you want to selectively append a file to a message or override the default signature file setting, use the /SIGNATURE_ FILE[=file-name] qualifier with the ANSWER, FORWARD, MAIL, REPLY, or SEND command.

Use the SHOW SIGNATURE_FILE command to show whether you have specified a default signature file. (The SHOW ALL command also displays signature file information.)

You can also set the default signature file at the DCL level by using the /SIGNATURE_FILE[=file-name] qualifier with the DCL command MAIL.

Note that when you create a mail message that includes a signature file, that message requires more temporary disk space than a conventional message because temporary files are created during the operation. After the message is sent, those temporary files are deleted.

When specifying the signature file name, also note the following:

- If you do not specify a file type, the default is .SIG.
- If you do not specify a directory, the Mail utility searches for the signature file in your mail directory.

6.7.9 Examples

- In the following example, the file BUSINESS_CARD.SIG is designated as the default file that will be automatically appended to every mail message sent using the FORWARD, MAIL, REPLY, or SEND command.

```
MAIL> SET SIGNATURE_FILE BUSINESS_CARD.SIG
```

- In the next example, the file GREETINGS.SIG is designated as the file that will be automatically appended to that specific reply instead of the default signature file.

```
MAIL> REPLY/SIGNATURE_FILE=GREETINGS.SIG
```

6.8 Other Ways to Send Messages

6.8.1 Replying to Messages

To reply to a message you have received, use the following procedure:

Step	Task
1	Type REPLY at the MAIL> prompt and press Return.
2	Enter your message and press Ctrl/Z to send the message or press Ctrl/C to quit.

6.8.2 Example

In the following example, a reply is being sent to STONE::THOMPSON. Note that after the reply command is entered, Mail automatically displays the To: and Subj: prompts:

```
To: STONE::THOMPSON
Subj:  RE: Budget Meeting
Enter your message below. Press CTRL/Z when complete. CTRL/C to quit:
```

6.8.3 Forwarding Messages

To forward a mail message to other users, enter the FORWARD command at the MAIL> prompt after you have read the message. Mail prompts you for the name of the addressee and a subject line. After you enter the requested information, press Return to send the message.

If you forward a message that consists of a .DDIF file, Mail sends the entire .DDIF file, including .DDIF semantics and the .DDIF tag, to the addressee.

6.8.4 Example

In the following example, a message is forwarded to user STONE::JONES:

```
MAIL> FORWARD Return
To: STONE::JONES Return
Subj: FYI -  Status of proposed budget meeting Return
```

6.9 Organizing Messages

6.9.1 Folders

To organize your mail messages, you can create your own mail files and folders. A mail file contains folders, and a folder contains mail messages. Each folder and file can contain any number of messages.

Typically, you organize your messages by creating folders rather than by creating mail files. As with the default mail folders (NEWMAIL, MAIL, WASTEBASKET), the folders you create are normally stored in the mail file MAIL.MAI. The name of the current folder is displayed in the top right corner of the screen each time you enter a READ or DIRECTORY command. You can work only with messages that are in your current folder.

If your mail file is very large (over 500 blocks), you might want to create separate mail files for the larger folders to improve Mail's performance.

6.9.2 Creating Mail Subdirectories

When you receive mail messages, they are written to files named MAIL$*xxxxxxxxxx*.MAI by default and are located in your top-level directory. (Note that the *x* characters represent a long, random file specification.) Your default mail file, MAIL.MAI, is created in your top-level directory the first time you receive a mail message.

To avoid the display of .MAI files in your top-level directory, use the Mail command SET MAIL_DIRECTORY. This command creates a mail subdirectory and moves all your .MAI files to that subdirectory. To move the .MAI files from a subdirectory back to your top level directory, use the SET NOMAIL_DIRECTORY command.

To display the name of the subdirectory that contains all your .MAI files, enter SHOW MAIL_DIRECTORY at the MAIL> prompt.

6.9.3 Example

In the following example, a user (FRED) creates the directory .MAIL:

```
MAIL> SET MAIL_DIRECTORY [.MAIL] Return
MAIL> SHOW MAIL_DIRECTORY Return
Your mail file directory is SY$LOGIN:[FRED.MAIL]
```

6.9.4 Moving Messages into Folders

You can use either the FILE command or the MOVE command to place the current message in a different folder. If the folder does not exist, Mail displays a message asking if you want to create it. After filing the message in the specified folder, Mail automatically deletes the message from the current folder.

6.9.5 Copying Messages

The Mail command COPY places a copy of the current message into the folder you specify. If the folder does not exist, Mail displays a message asking if you want to create it.

6.9.6 Example

In the following example, all messages containing the word MEETING are copied from the current folder to a folder named SCHEDULE. After the COPY command completes, there are two copies of each message, one in the current folder and one in the folder named SCHEDULE.

```
MAIL> SEARCH MEETING Return
MAIL> COPY SCHEDULE Return
Folder SCHEDULE does not exist.
Do you want to create it (Y/N, default is N)?Y  Return
%MAIL-I-NEWFOLDER, folder SCHEDULE created
```

The following command selects and displays the next message containing the word ‘‘meeting’’:

```
MAIL> SEARCH Return

MAIL> COPY SCHEDULE Return
MAIL> SEARCH Return
%MAIL-E-NOTFOUND, no messages containing 'MEETING' found
```

6.9.7 Selecting Folders

To display a list of the folders in your current mail file, enter the DIRECTORY/FOLDER command. To select a new folder as your current folder, use one of the following commands:

- SELECT *foldername*

  Selects the specified folder as the current folder. Subsequent Mail commands, such as READ and DIRECTORY, use the selected folder. You do not need to specify a folder name with each command.

- SET FOLDER *foldername*

  This command works the same as the SELECT command.

- DIRECTORY *foldername*

  Selects the specified folder as the current folder and lists the messages in the folder.

- READ *foldername*

  Selects the specified folder as the current folder and displays the specified message (by default, the first message in the folder).

6.9.8 Example

In the following example, the MEMOS folder is selected:

```
MAIL> DIRECTORY/FOLDER Return
Listing of folders in SYS$LOGIN:[FRED]MAIL.MAI;1
     Press CTRL/C to cancel listing
MAIL                                    MEETING_MINUTES
MEMOS                                   PROJECT_NOTES
STAFF

MAIL> SELECT MEMOS
```

6.9.9 Deleting Folders

To delete a mail folder, delete all the messages in the folder or move them to another folder. When you delete all messages in a folder, the empty folder is deleted automatically as soon as you select another folder.

6.9.10 Example

In the following example, the messages in the MUSIC folder are deleted:

```
MAIL> SELECT MUSIC Return
%MAIL-I-SELECTED, 2 messages selected
MAIL> DELETE/ALL Return
```

6.9.11 Creating and Accessing Mail Files

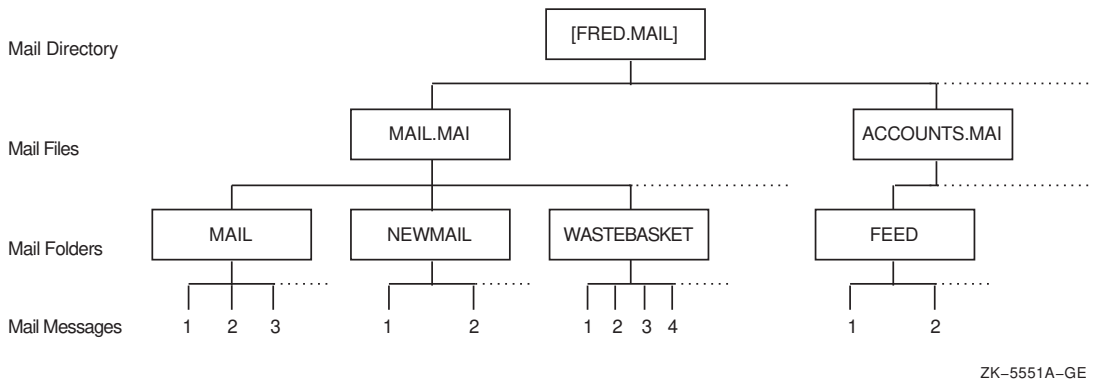
You can also create files to organize your mail messages. You use the same commands to create a mail file that you use to create a folder: COPY, MOVE, and FILE. After Mail prompts you for the name of the folder, it also prompts you for a file name. If you enter a new file name at the File: prompt, a new mail file is created.

To work within a mail file other than the default mail file, use the Mail command SET FILE to specify the alternate file. The Mail command SHOW FILE displays the name of the current mail file. When you change mail files, the WASTEBASKET folder of the current mail file is emptied and deleted (if AUTO_PURGE is set) and the mail file is closed.

6.9.12 Figure: Organization of a Typical Mail Directory

Figure 6–1 shows how a typical user might organize their mail.

Figure 6–1 Organizing Mail



6.9.13 Examples: Creating and Accessing Mail Files

- In the following example, the current message is moved into a folder named FEED in the ACCOUNTS file. The MOVE command creates the mail file ACCOUNTS.MAI, moves

the current message into the FEED folder, and deletes the message from its current folder and file.

```
MAIL> MOVE [Return]
_Folder:  FEED [Return]
_File:  ACCOUNTS [Return]
```

- In the following example, the FEED folder (which is in the ACCOUNTS file) is selected:

```
MAIL> SET FILE ACCOUNTS [Return]
MAIL> SET FOLDER FEED [Return]
MAIL> SHOW FILE [Return]
Your current mail file is SYS$LOGIN:[FRED.MAI]ACCOUNTS.MAI;1.
```

6.9.14 Correcting the Mail Message Count

If the number of new (unread) mail messages displayed on your screen is inconsistent with the actual number of new messages, enter the READ/NEW command when there is no new mail. You will know there is no new mail when you enter the READ/NEW command and receive one of the following system messages:

```
"%MAIL-W-NONEWMAIL, no new messages"

"%MAIL-E-NOMOREMSG, no more messages"
```

6.10 Deleting Messages

6.10.1 Using the DELETE Command

To delete a mail message from the current folder, either enter the DELETE command while you are reading the message or enter the DELETE command followed by the number (or range of numbers) of the message you want to delete. You can use either the hyphen (-) or the colon (:) to define the range of messages to be deleted.

6.10.2 Example

In the following example, messages 4, 5, 6, 11, 12, 14, 15, 16, and 17 are deleted:

```
MAIL> DELETE 4-6,11,12,14:17 [Return]
```

6.10.3 Recovering Deleted Messages

When you delete a message, the message is moved to a folder called WASTEBASKET. Deleted messages collect in the WASTEBASKET folder until you exit from the current mail file (either by exiting from Mail or by specifying a different mail file). If you have issued the SET AUTO_PURGE command, when you exit from the current mail file, WASTEBASKET is emptied and the folder itself is deleted. During your interactive Mail session, you can recover any deleted message by moving the message out of the wastebasket folder. You can also empty the WASTEBASKET folder by entering the PURGE command.

6.10.4 Example

In the following example, the mail message identified by the number 12 is deleted and then recovered from the WASTEBASKET folder.

```
MAIL> DELETE 12

MAIL> SELECT WASTEBASKET
%MAIL-I-SELECTED, 1 message selected

MAIL> DIRECTORY
    # top
From                 Date         Subject

    1 FABLES::WEST         11-DEC-1995  Meeting this week

MAIL> MOVE MAIL
```

6.11 Printing Mail Messages

6.11.1 Using the PRINT Command

To print a mail message, enter the PRINT command at the MAIL> prompt. By default, Mail sends your message to the SYS$PRINT queue. Mail files are not sent to a print queue until you press Ctrl/Z, enter the EXIT command, or enter the PRINT/PRINT command.

6.11.2 Specifying Print Queues

To specify a different queue, use the PRINT command qualifier /QUEUE. You can also select a different queue by issuing the SET QUEUE queue-name command; this queue will remain your default print queue until you enter another SET QUEUE command, even if you exit Mail.

6.11.3 Examples

- In the following example, the mail message is submitted to the AK34$PRINT print queue:

  ```
  MAIL> PRINT/QUEUE=AK34$PRINT Return
  ```

- In the following example, the default print queue is changed from SYS$PRINT to AK34$PRINT:

  ```
  MAIL> SET QUEUE AK34$PRINT Return
  ```

6.12 Protecting Mail Files

6.12.1 Default Protection

Mail files (for example, MAIL.MAI) are protected so that no one else can read them and so that you cannot accidentally delete them. The protection code that Mail gives .MAI files is: (S:RW,O:RW,G:,W:). The system (including Mail itself) and the owner (you) can read and write to the file. The group and world are denied all access.

The Mail utility also has default file protection to discourage mail tampering. However, Mail is not completely secure from tampering. Anyone with sufficient privileges can change protection and access mail files.

6.12.2 Security Measures

Mail files are within your own directory, so you have the option of applying the file protection techniques for sensitive files described in Chapter 19. In addition:

- Use your judgment in responding to mail requests: if a node is outside your local VMScluster environment, it is possible that the source node is incorrectly identified, either accidentally or intentionally.
- It is best to use discretion in the content of your mail messages and in the selection of your audience.
- Never reveal your password or send details about how to use your account. You have no control over information in a mail message once you have sent it.

6.13 Using Text Editors in Mail

6.13.1 Using EVE

You can use a text editor to write a message before you send it. To do so, specify the /EDIT qualifier with the SEND command. After you respond to the To: and Subj: prompts, Mail invokes the text editor. Unless you have selected a different editor, Mail invokes the DECTPU-based EVE editor.

The [End of file] marker moves down as you enter text. For more information about the EVE editor, see Chapter 8. To send the message, press the Do key and enter the EXIT command. To cancel the send operation, press the Do key and enter the QUIT command.

6.13.2 Example

In the following example, EVE is used to create a mail message:

```
MAIL> SEND/EDIT Return
[End of file]

Buffer:  MAIN                    | Write | Insert | Forward
```

6.13.3 Note: DDIF Files

Do not edit a .DDIF mail file because you will no longer be able to use the file as a .DDIF file. If you edit a .DDIF mail file, you can access only the text of the file.

6.13.4 Using /EDIT Qualifier Keywords

By specifying the /EDIT qualifier when you invoke Mail, you can use the editor for sending, replying, and forwarding during the ensuing mail session. You can also use keywords with the /EDIT qualifier to set the default for Mail.

To invoke the editor only when you are replying to a message, use the REPLY keyword with the MAIL/EDIT command. To invoke the editor and display the message to which you are replying, use the REPLY keyword with the =EXTRACT option. If you do not specify a keyword with /EDIT, the default is /EDIT=(SEND,REPLY).

To send or reply to a message, EXIT from the editor. To cancel a SEND or REPLY command, enter the QUIT command to exit from the editor.

6.13.5 Examples

- In the following example, the editor will be invoked for every mail message that is sent or forwarded:

  ```
  $ MAIL/EDIT=(SEND,FORWARD) Return
  ```

- In the following example, the editor will be invoked for every message that is replied to:

  ```
  $ MAIL/EDIT=(REPLY) Return
  ```

- In the following example, the editor will be invoked and the message to which you are replying will be included as text every time you reply to a message:

  ```
  $ MAIL/EDIT=(REPLY=EXTRACT) Return
  ```

6.13.6 Default Editor

By default, Mail invokes the DECTPU-based EVE editor when you specify the Mail command SEND/EDIT. If you are not running an ANSI-compliant CRT (hardcopy) terminal, you will not be able to invoke DECTPU-based editors (such as EVE and LSE). If you attempt to invoke a DECTPU-based editor, you will receive the error message :

```
%TPU-E-NONANSICRT, SYS$INPUT must be supported CRT
```

6.13.7 Selecting an Editor

By entering the Mail command SET EDITOR, you can specify that a different editor be invoked instead of EVE. For example, to select the EDT editor, issue the Mail command SET EDITOR EDT. The EDT editor remains your default Mail editor (even if you log out of the system and log back in) until you enter another SET EDITOR command.

6.13.8 Displaying the Selected Editor

To display the name of the selected Mail editor, enter the Mail command SHOW EDITOR.

6.13.9 Example

In the following example, the EDT editor is selected and then used to create a message:

```
MAIL> SET EDITOR EDT Return
MAIL> SHOW EDITOR Return
Your editor is EDT.

MAIL> SEND/EDIT Return
To:   STONE::THOMPSON Return
Subj: Budget Meeting Return
[EOB]
*
```

6.13.10 Using the EDT Editor

When EDT is used to edit messages, you will see an asterisk (*) after you enter the subject line and press Return. Enter the CHANGE command and press Return to switch to the screen editor. For more information on using EDT, see Chapter 9. Enter the text of your message using EDT commands to move around in the buffer. A **buffer** is a temporary storage area that exists only during an editing session. To send the message, press Ctrl/Z. To cancel and not send the message, press Ctrl/C.

6.13.11 Using a Command File to Edit Mail

You can define the logical name MAIL$EDIT to be a command file before entering Mail. Then, when you issue any Mail command that invokes an editor, the command file will be called to perform the edit. In the command file, you can also invoke other utilities such as the spell-checker and you can specify any function that can be done in a command file. Refer to Appendix C for an annotated example of a MAILEDIT.COM command procedure and refer to Chapter 15 and Chapter 16 for more information on command files.

6.13.12 Overriding Your Selected Editor

If you wish to temporarily override your selected editor, you can define MAIL$EDIT to be the string "CALLABLE_" with the desired editor name appended. For example, to use callable EDT rather than callable EVE, you can type the following command:

```
$ DEFINE MAIL$EDIT CALLABLE_EDT
```

If you issue the SET EDITOR command during a session that was invoked with MAIL$EDIT defined, you override both your permanent selected editor and the current editor setting. In order to use the command file defined by MAIL$EDIT again, you must exit from Mail and restart it.

6.14 Customizing Your Mail Environment

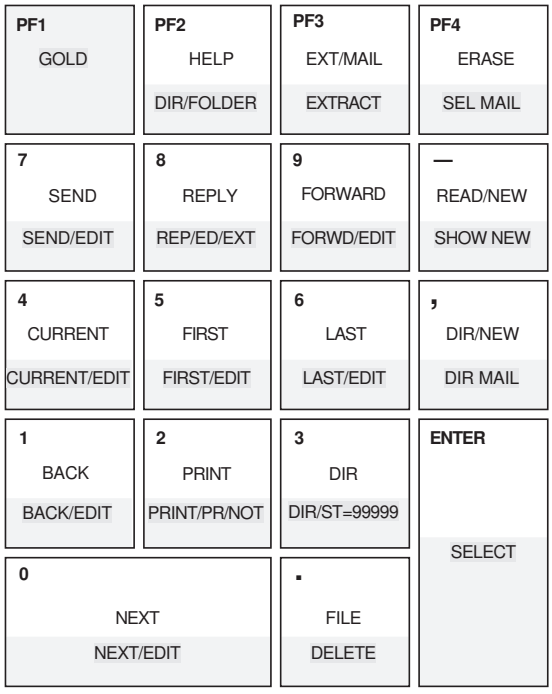
6.14.1 Using the Mail Keypad

You can use the numeric keypad on your keyboard to execute commands in Mail. Most keypad keys can execute two commands.

6.14.2 Mail Utility Keypad

Figure 6–2 shows the Mail keypad. To enter the top command for each key shown, press the appropriate key. To enter the bottom command shown, press the PF1 key first, and then the desired function key.

Figure 6–2 Mail Utility Keypad



ZK–1744–GE

6.14.3 Example of Using the Keypad

To execute the Mail command SEND, press KP7. To execute the Mail command SEND/EDIT, press the PF1 key first and then press KP7.

6.14.4 Redefining Keypad Keys

You can redefine the keypad keys to execute Mail commands when you are in Mail. Note that the previous definition of the key is superseded when you redefine a key.

Defining keypad keys in Mail is similar to defining keypad keys to execute DCL commands.

6.14.5 Example

In the following example, the key KP2 is defined as the Mail command PRINT/PARAM=PAGE_ORIENT=LANDSCAPE. After KP2 is defined, you can press it to display the PRINT/PARAM=PAGE_ORIENT=LANDSCAPE command:

```
MAIL> DEFINE/KEY KP2 "PRINT/PARAM=PAGE_ORIENT=LANDSCAPE" Return
```

6.14.6 Assigning Additional Definitions

To increase the number of key definitions available on your terminal, use the /STATE qualifier. You can assign many definitions to the same key as long as each definition is associated with a different state. State names can be any alphanumeric string. By specifying states, you can press a key once to enter a command and a second time to enter a qualifier.

6.14.7 Example

In the following example, PF1 (pressed twice) is defined as DIRECTORY/FOLDER:

```
MAIL> DEFINE/KEY PF1 "DIRECTORY"/SET_STATE=FOLDER /NOTERMINATE Return
MAIL> DEFINE/KEY PF1 "/FOLDER" /IF_STATE=FOLDER /TERMINATE Return
```

Press PF1 twice to enter the command DIRECTORY/FOLDER. The /TERMINATE qualifier ends the command line so you do not need to press the Return key.

6.14.8 Creating Permanent Key Definitions

Any keypad keys that you define during a Mail session are lost when you exit from Mail. To retain keypad key definitions from one Mail session to another, create a file containing key definitions (for example, MAIL$KEYDEF.INI) in your top-level directory. For example, the following MAIL$KEYDEF.INI file contains six key definitions:

```
DEFINE/KEY PF1 "DIRECTORY "     /NOTERMINATE    /SET_STATE=folder
DEFINE/KEY PF1 "/FOLDER"        /TERMINATE      /IF_STATE=folder
DEFINE/KEY PF2 "SELECT "        /NOTERMINATE    /SET_STATE=mail
DEFINE/KEY PF2 "MAIL"           /TERMINATE      /IF_STATE=mail
DEFINE/KEY PERIOD "READ "       /NOTERMINATE    /SET_STATE=new
DEFINE/KEY PERIOD "/NEW"        /TERMINATE      /IF_STATE=new
```

To execute these commands each time you invoke Mail, enter the following command line in your login command file (LOGIN.COM):

```
$ DEFINE MAIL$INIT SYS$LOGIN:MAIL$KEYDEF.INI
```

6.15 Summary of Mail Commands

6.15.1 Overview

This section contains a summary of all Mail utility commands. For complete information on qualifiers used with these commands, refer to online help.

6.15.2 Reading Messages

Use the following commands to read messages:

- **BACK**

  Displays the message preceding the current or last-read message when the last command issued was READ. When the last command issued was DIRECTORY, the BACK command displays the preceding screen of the directory listing.

- **CURRENT**

  Displays the beginning of the message you are currently reading.

- **DIRECTORY** [folder-name]

  Displays a list of the messages in the current mail file, including message number, sender's name, date, and subject.

- **ERASE**

  Clears your terminal screen.

- **EXTRACT**

  Places a copy of the current message into the specified output file. To copy a mail message to a folder in a Mail file, use either the COPY, FILE, or MOVE command.

- **FIRST**

  Displays the first message in the current folder.

- **LAST**

  Displays the last message in the current folder.

- **NEXT**

  Skips to the next message and displays it.

- **READ** [folder-name] [message-number]

  Displays your messages. Pressing the Return key is the same as entering the READ command without parameters.

- **SEARCH** search-string

  Searches the currently selected folder for the message containing the first occurrence of the specified text string.

- **SHOW NEW_MAIL_COUNT**

  Displays the number of unread mail messages.

6.15.3 Exchanging Messages

Use the following commands to exchange messages:

- **ANSWER** [filespec]
  REPLY [filespec]

  Sends a message to the sender of the message you are currently reading or of the one you last read.

- **FORWARD**

  Sends a copy of the message you are currently reading (or have just read) to one or more users.

- **MAIL** [filespec]
  SEND [filespec]

  Sends a message to one or more users.

6.15.4 Removing Messages

Use the following commands to remove messages:

- **DELETE** [message-number]

  Deletes either the message you are currently reading, a range of messages, or the message you just read, and moves it to the WASTEBASKET folder.

- **PURGE**

  Deletes all the messages in the WASTEBASKET folder. When you exit from Mail or enter a SET FILE command (to select a new mail file), an implicit purge is done to empty the WASTEBASKET folder, unless you have previously entered the SET NOAUTO_PURGE command.

- **SET [NO]AUTO_PURGE**

  Determines whether Mail empties the WASTEBASKET folder when you enter the EXIT or SET FILE command. When you use the SET NOAUTO_PURGE command, you must enter the PURGE command periodically to delete the messages in the WASTEBASKET folder.

- **SHOW AUTO_PURGE**

  Displays whether messages in the WASTEBASKET folder are deleted (purged automatically) when you enter the EXIT or SET FILE command.

6.15.5 Printing Messages

Use the following commands to print messages:

- **PRINT**

  Adds a copy of the message you are currently reading to the print queue. The files created by the PRINT command are released to the print queue when you exit from Mail. Multiple messages are concatenated into one print job unless you use the /NOW or /PRINT qualifier.

- **SET [NO]FORM** form-name
  SHOW FORM

  Sets the default print form for printing done within Mail. The SET NOFORM command clears the default print form. The SHOW FORM command displays the default print form.

- **SET [NO]QUEUE** queue-name
  SHOW QUEUE

  Sets the default print queue to be used when you enter the PRINT command from within Mail. SET NOQUEUE clears the previously defined print queue and sets the queue to SYS$PRINT, the default print queue. The SHOW QUEUE command displays your default print queue.

6.15.6 Organizing Messages

Use the following commands to organize messages:

- **COPY** folder-name [filename]

  Copies a message to another folder without deleting it from the current folder. If the specified folder does not exist, it is created.

- **FILE** folder-name [filename]
  MOVE folder-name [filename]

  Moves the current message to the specified folder and deletes the message from the original folder.

- **SELECT** [folder-name]
  SET FOLDER [folder-name]

  Establishes a set of messages that you can affect as a group. You can copy or move this set of messages from one folder to another. You can also read and delete, or search and extract a set of messages. In addition, you can use the SELECT and SET FOLDER commands to move from one folder to another.

- **SET FILE** filename

  Establishes (or opens) another file as the current mail file. By default, your mail file is MAIL.MAI. If you use the COPY command, the FILE command, or the MOVE command to create other mail files (for example, JOKES.MAI or HISTORY.MAI), you can then use the SET FILE command to open the Mail files.

- **SHOW FILE**

  Displays the name of the mail file that is currently open.

- **SHOW FOLDER** [folder-name]

  Displays the current folder name.

- **SET WASTEBASKET_NAME** folder-name

  Changes the name of the WASTEBASKET folder, which contains messages to be deleted. You can delete all the messages in the WASTEBASKET folder by entering the PURGE command. If AUTO_PURGE is set, when you enter the EXIT command, messages in the WASTEBASKET folder will be deleted. If AUTO_PURGE is set, you can avoid deleting messages in the WASTEBASKET folder by entering the QUIT command.

- **SHOW WASTEBASKET_NAME**

  Displays the name of the WASTEBASKET folder.

- **SHOW DELETED**

  Displays the amount of deleted message space in the current mail file.

6.15.7 Marking Messages

The following commands are used for marking messages:

- **MARK** [message-number]

  Sets the current or specified message as marked. Marked messages are displayed with an asterisk (*) in the left-hand column of the directory listing. To select or organize marked messages, use the SELECT command with the /MARKED qualifier.

- **UNMARK** [message-number]

  Sets the current or specified message as unmarked. The asterisk (*) in the left-hand column of the directory listing is deleted.

6.15.8 Customizing the Mail Environment

The following commands are used for customizing the mail environment:

- **DEFINE/KEY** key-name string

  Defines a key to execute a Mail command. You can press the key to enter a command instead of typing the command name.

- **SHOW KEY** [key-name]

  Displays the key definitions created by the DEFINE/KEY command.

- **EDIT** [filename]

  Invokes your selected editor and enables you to edit a message before you send it.

- **HELP** [topic]

  Displays information about Mail. To obtain information about individual commands or topics, enter HELP followed by the command or topic name.

- **SET [NO]CC_PROMPT**

  Sets the default for determining whether the carbon copy (CC:) prompt appears when sending a message.

- **SET COPY_SELF** command[,command]

  Sets the default for determining whether the SEND, REPLY, or FORWARD commands return to the sender a copy of the message being sent.

- **SHOW COPY_SELF**

  Displays which command (SEND, REPLY, or FORWARD) automatically sends a copy of the message to you.

- **SET [NO]SIGNATURE_FILE**

  Sets the Mail utility to append a signature text file to the end of a mail message automatically whenever you use the ANSWER, FORWARD, MAIL, REPLY, or SEND command.

- **SHOW SIGNATURE_FILE**

  Displays information that shows whether you have specified a default signature file and, if so, the name of that file. (The SHOW ALL command also displays signature file information.)

- **SET EDITOR** editor-name

  Selects the text editor to be used when you edit a message (for example, with the Mail command SEND/EDIT). You can use any callable editor available on your system. This command overrides any definition that the command procedure MAIL$EDIT has set.

- **SHOW EDITOR**

  Displays the name of your selected text editor.

- **SET [NO]FORWARD** address

  Sets a forwarding address for your mail.

- **SHOW FORWARD**

  Displays the name of your current forwarding address.

- **SET [NO]MAIL_DIRECTORY** [.subdirectory-name]

  Specifies that all mail files (file type .MAI) be moved from your SYS$LOGIN directory to the specified subdirectory.

- **SHOW MAIL_DIRECTORY**

  Displays the name of the device and directory containing all your .MAI files.

- **SET [NO]PERSONAL_NAME** ‘‘text-string’’

  Appends a text string to the end of the From: field of mail messages you send. You can fill this field with your full name or any other information. Note that your personal name must begin with a letter and may not have two consecutive spaces.

- **SHOW PERSONAL_NAME**

  Displays the text string established with the SET PERSONAL_NAME command.

- **SHOW ALL**

  Displays detailed information about your current Mail settings.

6.15.9 Exiting or Transferring Control

The following commands are used for exiting Mail or transferring control:

- **ATTACH** [process-name]

  Permits you to switch control of your terminal from your current process to another process in your job. For example, while you are editing a file, you can use the SPAWN command to move to a subprocess (Mail) to read a new mail message. Then, you can enter the ATTACH command to move back to the editing session.

- **EXIT**

  Exits from Mail. When you enter the EXIT command, any messages in the WASTEBASKET folder are deleted unless you have issued the command SET NOAUTO_PURGE. You can also exit from Mail by pressing Ctrl/Z.

- **QUIT**

  Exits from Mail without emptying the WASTEBASKET folder (deleted messages are not destroyed unless you enter the EXIT command or press Ctrl/Z). The QUIT command performs the same function as Ctrl/Y.

- **SPAWN** [command]

  Creates a subprocess of the current process. You can use the SPAWN command to leave Mail temporarily, perform other functions (such as displaying a directory listing or printing a file), and then return to Mail.

6.15.10 Mail File Compression

The following command is used for compressing mail files:

- **COMPRESS** [filespec]

  Makes an indexed mail file smaller. If you do not specify a file name, Mail compresses the mail file that is currently open. If there is no open mail file, Mail compresses the default mail file (MAIL.MAI).

6.15.11 System Management Commands

The following commands are used for system management:

- **REMOVE** username

  Removes a user record from the system's mail profile, data file SYS$SYSTEM:VMSMAIL_PROFILE.DATA. Requires SYSPRV privileges.

- **SHOW FORWARD**

  Displays the name of a user's current forwarding address.

- **SHOW PERSONAL_NAME**

  Displays the text string that a user has established with the SET PERSONAL_NAME command.

7

Phone: Communicating with Other Users

7.1 Overview

The OpenVMS Phone utility (PHONE) is a communication program designed to allow users to ‘‘talk’’ to each other via their terminals, computers, or computer networks. This chapter includes information on:

- Using Phone
- Entering Phone commands
- Customizing your Phone viewport
- Summary of Phone commands

7.1.1 References

For additional information about the commands described in this chapter, refer to online help.

7.2 Using Phone

7.2.1 Phone Features

Phone (also sometimes referred to as the Phone facility) is designed to provide features similar to that of actual telephone communications, such as the hold button, conference calls, and telephone directories. You can use Phone to communicate with other users on your system or with any other system connected to your system by DECnet for OpenVMS networks.

7.2.2 Invoking Phone

To invoke Phone, enter the PHONE command at the DCL prompt and press Return. You can specify the user name of the person with whom you want to communicate before or after you enter Phone. When you invoke the Phone utility, Phone takes control of your terminal and displays the Phone viewport.

7.2.3 Example of a Phone Viewport

The following figure shows the Phone viewport:

```
$ PHONE    ❶

                    OpenVMS Phone Facility          11-DEC-1995

% ❷

------------------------------------------------------------
                      TAURUS::SMITH  ❸




------------------------------------------------------------
                     GEMINI::PETERS  ❹



------------------------------------------------------------
```

The fields on the viewport are as follows:

❶ The command that is used to enter the Phone utility.

❷ The Phone prompt, also known as the switchhook. This is where you enter Phone commands. You can get to the Phone command line by pressing the switchhook (%) character at any time during your conversation. The switchhook character is the percent sign (%) on your keyboard.

❸ The top part of the viewport is where the conversation text that you type appears.

❹ The lower part of the viewport is where the conversation text that other participants type appears. You can have up to six participants in a phone conversation.

7.2.4 Help on Phone

You can obtain information about Phone by invoking the HELP command at the DCL prompt as follows:

```
$ HELP
Topic? PHONE
```

You can also enter Help from within the Phone utility by entering the HELP command at the switchhook (%) prompt.

7.2.5 Viewport

Each person engaged in the conversation has a viewport on the screen. Phone can display as many as six viewports at a time. The viewport contains information regarding the user's name, the text of the conversation, and various status indicators, such as who is on hold. User names of people that you have on hold can be temporarily eliminated from the screen to make room for new participants.

7.3 Entering Phone Commands

7.3.1 Switchhook

To enter Phone commands, you must first press the switchhook character (%). If you are using the Phone utility but are not currently engaged in a conversation, the switchhook character is optional because there is no ambiguity between a command and conversation.

7.3.2 Refreshing the Screen

You can press Ctrl/W at any time during your current conversation to refresh the screen.

7.3.3 Logical Names

The DIAL, DIRECTORY, MAIL, and PHONE commands accept logical names. To prevent Phone from treating a parameter to these commands as a logical name, prefix the parameter with an underscore.

7.3.4 Special Characters

When you are engaged in a conversation, most of the characters that you type are considered part of that conversation and are sent to each participant. The exception is the percent sign (%), which signals that you want to enter a Phone utility command. You can enter any Phone utility command during a conversation. See Section 7.5 for a list of available Phone commands.

7.4 Customizing Your Phone Viewport

7.4.1 Using PHONE Command Qualifiers

When entering the DCL command PHONE, you can supply the following qualifiers that modify the characteristics of the simulated telephone:

/SCROLL	Determines how new lines of text are displayed on the screen when the viewport becomes full.
/SWITCH_HOOK	Specifies the character to be used for the switchhook prompt. The switchhook character must be entered before each Phone utility command that is entered during a conversation.
/VIEWPORT_SIZE	Specifies the maximum number of lines in a viewport, including the heading line and the bottom line of dashes.

7.4.2 For Additional Information

For complete descriptions of Phone qualifiers, enter HELP PHONE at the DCL prompt.

7.5 Summary of Phone Commands

7.5.1 Phone Commands

The following table summarizes all of the Phone commands available. There are no qualifiers associated with the commands in this table.

Command	Description
ANSWER	Answers the phone when you receive a call.
DIAL	Places a call to another user.
DIRECTORY	Displays a list of those users with whom you can talk on your system or on any other system in the network.
EXIT	Exits from the Phone utility.
FACSIMILE	Allows you to include the contents of a file in your conversation.
HANGUP	Hangs up your phone. This disconnects all current links—the current conversation, anyone you have on hold, and anyone who has you on hold.
HELP	Enables you to obtain online information about the Phone utility.
HOLD	Enables you to put on hold other users who are currently participating in a conversation with you.
MAIL	Allows you to send a brief message to another person.
PHONE	Is synonymous with the DIAL command.
REJECT	Allows you to reject a call from another user while you are using Phone.
UNHOLD	Reverses the most recently entered HOLD command.

7.5.2 For Additional Information

For complete descriptions of Phone commands, invoke Phone and enter the HELP command.

8
Editing Text Files: Using EVE

8.1 Overview

The Extensible Versatile Editor (EVE) is a general-purpose text editor based on the DEC Text Processing Utility (DECTPU). This chapter includes information on:

- EVE features
- Getting help
- Beginning an editing session
- Entering commands
- Saving your edits and exiting from EVE
- Moving the cursor
- Entering text
- Erasing and restoring text
- Moving text
- Copying text
- Box editing
- Using pending delete
- Finding and replacing text
- Using command line qualifiers
- Alternate methods to invoke EVE
- Journaling and recovery
- EVE formatting commands
- Using buffers
- Creating a subprocess

8.1.1 Conventions

In this chapter, EVE key names are shown (with the SHOW KEY or HELP KEYS command) by using a slash for control keys, shifted function keys, and Alt key combinations, and a space or a dash for GOLD key sequences. Thus, key combinations that require you to hold down one key (such as Ctrl) while pressing another key are shown with a slash; key combinations in which you press one key after another (such as GOLD-Help) are shown with a space or a dash.

8.1.2 References

For additional information on EVE, refer to:

- Online help in EVE
- The *Extensible Versatile Editor Reference Manual*

8.2 EVE Features

8.2.1 Overview

DECTPU is a high-performance, programmable text processor. With EVE software, you can create and edit text files such as business letters, technical documents, and program source files.

EVE is the default editor on the OpenVMS operating system. Unless you define a different default editor, EVE is invoked when you enter the EDIT command.

You can use EVE on a character-cell terminal (VT100, VT200, VT300, or VT400 series) or on a workstation with the OpenVMS DECwindows Motif user interface.

8.2.2 EVE Usage

With EVE, you can do the following:

- Create and edit text files such as letters, reports, program sources, and other documents.
- Perform text formatting operations, such as erase, cut, paste, fill, find, replace, and paginate.
- Use multiple buffers and windows to view and edit different files in the same editing session.
- Define keys for editing operations, including learn sequences (to bind several commands or keystrokes to a single key) and setting the EDT keypad or WPS–PLUS keypad.
- Select text in boxes or in linear ranges for cut-and-paste or other edits.
- Use wildcards to search for patterns of text.
- Execute DCL commands (such as DIRECTORY) from within the editor.
- Run DECspell to check a selection or an entire buffer.
- Spawn subprocesses or attach to other processes.
- Compile and execute DECTPU procedures to extend EVE.
- Add to or delete menu items from the DECwindows interface.
- Save compiled procedures, menu definitions, key definitions, and other customizations for future sessions. (See Appendix A for more information on customizing EVE.)
- Use initialization files at startup or during an editing session.
- Recover your work by using keystroke or buffer-change journaling when a system failure interrupts your editing session.

- Get comprehensive online help on EVE commands, keys, menu items, and other topics, including DECTPU built-in procedures.

8.2.3 Modifying Files

Once you know how to invoke EVE and how to enter commands, you can use EVE commands to create and edit files. Using editing keys and commands, you can move the cursor, set buffer mode, and perform editing operations such as entering, erasing, restoring, and moving text.

8.3 Getting Help

You can get online help at any time during your editing session. There are two kinds of online help available with the EVE editor:

- Keypad help, which you access with the Help key on your terminal
- EVE help, which you access with the HELP command at the EVE command prompt

8.3.1 Using Keypad Help

To access keypad help, follow these steps:

1. Press the Help key.

   The Help utility displays a diagram of your keypad.
2. Follow the directions on the screen to get information on:
   - EVE commands

     To get help on EVE commands, enter a command name or a question mark (?) and press the Return key.
   - Defined keys

     To get help on a key that you have defined, press that key, or use the SHOW KEY command.
   - Listing key definitions

     To list all key definitions, type the word *keys* and press the Return key, or press GOLD HELP. The GOLD key is the PF1 key on the numeric keypad.
3. Press the Return key to exit from Help.

8.3.2 Using EVE Help

To use the HELP command to access EVE Help, follow these steps:

1. Press the Do key.
2. Enter the command HELP.

   Use the Prev Screen and Next Screen keys to scroll through the list of available help topics.
3. Press the Return key to exit from Help.

To get information about a particular command, enter HELP followed by the command name and press the Return key. The help text appears on the screen. You can also get help on DECTPU built-in procedures by entering the command HELP TPU.

8.3.3 Example

The following example shows the help text for the MOVE BY LINE command:

```
MOVE BY LINE

Moves the cursor a line at a time in the current direction.

Keys:  EVE Default                        VT100 Keypad
       ------------------------------------------------
       F12                                MINUS on keypad

Steps:

   1.  If necessary, set the direction to move in --- forward or reverse.

   2.  Use MOVE BY LINE (see key list above).

Usage notes:

o  In forward direction, moves to the end of the current line, or to the
   end of the next line, if any.

o  In reverse direction, moves to the start of the current line, or to
   the start of the next line, if any.

Related topics:

   CHANGE DIRECTION      END OF LINE      LINE      START OF LINE
```

8.4 Beginning an Editing Session

8.4.1 Invoking EVE

To invoke EVE, use the EDIT/TPU command. When you begin an editing session, you can specify the name of an existing file or a new file you want to create. If you do not specify a file name now, EVE prompts you for a file name when you end your editing session if you have added text to the default buffer called Main.

8.4.2 Example: Invoking EVE

The following example invokes EVE to create a new file named NEWFILE.DAT:

```
$ EDIT/TPU NEWFILE.DAT

[End of file]❶

                                        ❷


Buffer: NEWFILE.DAT              | Write | Insert | Forward ❸
Command: ❹
Editing new file. Could not find: FABLES.TXT ❺
```

As you examine the EVE screen display, note the following:

❶ The end-of-file marker marks the end of an EVE buffer. It is visible only on the screen and does not become part of your file. When you add text to the buffer, the end-of-file marker moves down. Depending on the length of your terminal screen, the marker may not be visible when you view the beginning of a buffer that contains many lines of text.

❷ A window is an area of your screen that displays a buffer. EVE buffers exist only during the editing session. When you end an editing session, you can save your edits or discard them.

❸ A highlighted status line appears at the bottom of the EVE window and provides information about the buffer you are viewing in the window. The status line shows the buffer name, editing status (write or read-only), current mode (insert or overstrike), and current direction (forward or reverse).

❹ You use the command line to enter line-mode commands (see Section 8.5). You get the command line by pressing the Do key.

❺ The message window contains an informational message that appears beneath the highlighted status line when you invoke EVE and specify a file name on the command line. The message states either that the file is a new file or that a certain number of lines were read from an existing file. During the editing session, EVE displays other messages in the message window.

8.5 Entering Commands

8.5.1 Overview

There are two ways to enter EVE commands:

- Type in commands on the command line interface
- Use defined keys on either the EDT or WPS keypad

8.5.2 Typing Commands

To type a command, follow these steps:

1. Press the Do key.

   The cursor moves to the command window and EVE prompts you to type a command.

2. Type a command. You can abbreviate the command by using the first few letters of the command. EVE is not case sensitive. You can use any combination of uppercase and lowercase characters in the command line except when specifying strings for the FIND and REPLACE commands.

3. Press the Do key or the Return key.

EVE executes the command or prompts you for further information.

8.5.3 Using Defined Keys

You can use defined keys to enter EVE commands. Each defined key performs one editing command. You can also define your own keys to perform EVE functions (see Section A.2).

EVE defines some keys by default. The predefined keys on VT200, VT300, and VT400 series terminals include:

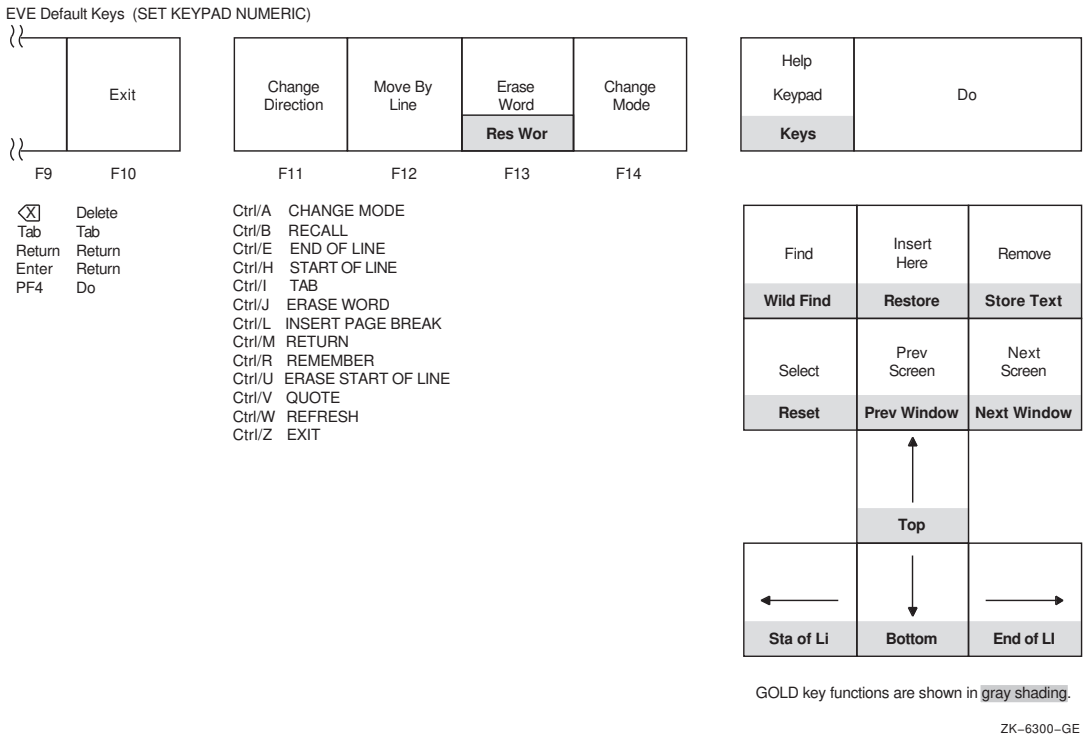
- The minikeypad (located between the main keyboard keys and the numeric keypad, above the arrow keys)
- Certain **function keys**
- Certain control key sequences

Control keys, arrow keys, and the Tab, Return, and Delete keys have the same definitions on all three types of terminal.

8.5.4 Figure: EVE Keys

Figure 8–1 shows the predefined keys for the VT200, VT300, and VT400 series terminal.

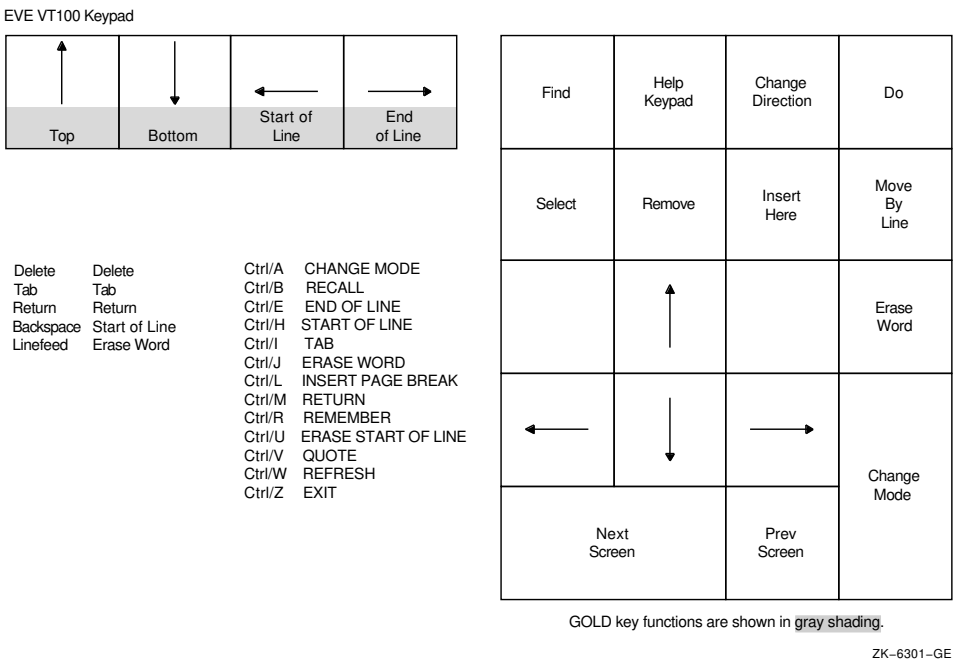
Figure 8–1 EVE Keys—VT200, VT300, and VT400 Series Terminals



8.5.5 Figure: EVE Keys—VT100 Series Terminals

On VT100 series terminals, EVE automatically defines most of the numeric keypad keys, the four arrow keys, and certain control keys. Figure 8–2 shows the predefined keys for the VT100 series terminal.

Figure 8–2 EVE Keys—VT100 Series Terminals



8.6 Saving Your Edits and Exiting from EVE

8.6.1 Overview

You can use one of the following methods to save your edits:

- WRITE FILE command

  Saves a file without terminating your editing session

- EXIT command

  Terminates your editing session and saves the changes to your file

- QUIT command

  Terminates your editing session without saving changes to your file

8.6.2 Using the WRITE FILE Command

To save the text in your buffer by writing it to a file without exiting from EVE, use the WRITE FILE command. If no file is associated with your buffer, EVE prompts for a file name, as follows:

```
Type filename for buffer Main (press RETURN to not write it):
```

Type the name of the file and press the Return key to write out the buffer to a file.

8.6.3 Using the EXIT Command

To save your edited text, use the EXIT command. You can enter the EXIT command by pressing the F10 key (on VT200, VT300, or VT400 series terminals) or by pressing Ctrl/Z.

If you have modified the current buffer, EVE creates a new version of the file with the same file name and file type as the original version, with the version number incremented by 1. For example, if you use the EXIT command after modifying a file named FUN.DAT;1, the output file is named FUN.DAT;2.

8.6.4 Using the QUIT Command

To end a session without saving your edits, enter the QUIT command. Type YES (Y) and press the Return key if you want to quit without saving the edits. If you change your mind and decide to save your edits, type N, press the Return key, and use the EXIT command to exit from the buffer.

If you have modified buffers other than the current one, EVE asks if you want to save the contents of the other buffers. If you type Y, EVE creates a new version of an existing file, incrementing the version number by 1. EVE prompts for a file name if no file currently exists.

If no buffers have been modified, then EXIT and QUIT are the same. For example, if you use EVE to inspect a file without making edits, you can quit by pressing Ctrl/Z.

8.6.5 Example

In the following example, there is a modified a buffer named FUN.DAT and the QUIT command is entered:

```
Command: QUIT
Buffer modifications will not be saved, continue quitting (Y or N)?
```

8.7 Moving the Cursor

8.7.1 Overview

When editing files with EVE, you move the cursor where you want to perform an editing function. The more quickly and efficiently you move the cursor through the text, the more time you save in your editing session. You can use the keyboard or commands to move the cursor.

8.7.2 EVE Editing Keys That Move the Cursor

The following table shows EVE editing keys that move the cursor. For more information about the GOLD key combinations listed, see the online help topic GOLD.

Key or Key Sequence	Function
↑	Same as MOVE UP. Moves the cursor up one line. On VT100 series terminals, KP5 is also defined as MOVE UP.
↓	Same as MOVE DOWN. Moves the cursor down one line. On VT100 series terminals, KP2 is also defined as MOVE DOWN.
←	Same as MOVE LEFT. Moves the cursor one character or column to the left. On VT100 series terminals, KP1 is also defined as MOVE LEFT.
→	Same as MOVE RIGHT. Moves the cursor one character or column to the right. On VT100 series terminals, KP3 is also defined as MOVE RIGHT.
Ctrl/E or GOLD →	Same as END OF LINE. Moves the cursor to the end of the current line.
Ctrl/H or GOLD ←	Same as START OF LINE. Moves the cursor to the beginning of the current line.
GOLD ↑	Same as TOP. Moves the cursor to the top of the current buffer.
GOLD ↓	Same as BOTTOM. Moves the cursor to the bottom of the current buffer.
GOLD Next Screen	Same as NEXT WINDOW. Moves the cursor to the last position the cursor occupied in the next window on your screen, if you are using two or more windows.
GOLD Prev Screen	Same as PREVIOUS WINDOW. Moves the cursor to the last position the cursor occupied in the previous window on your screen, if you are using two or more windows.

8.7.3 EVE Commands That Move the Cursor

Table 8–1 shows EVE commands that move the cursor.

Table 8–1 EVE Commands That Move the Cursor

Command	Function
BOTTOM	Moves the cursor to the end of the current buffer. By default, EVE defines GOLD ↓ as BOTTOM.
CHANGE DIRECTION	Changes the direction of the current buffer. The direction of the buffer is shown in the status line.
END OF LINE	Moves the cursor to the end of the current line. By default, EVE defines both Ctrl/E and GOLD → as END OF LINE.
FORWARD	Default setting. Sets the direction of the current buffer to forward; that is, to the right and down. The direction of the buffer is shown in the status line.
GO TO	Moves the cursor to the position you specify, as previously labeled with the MARK command.
LINE	Moves the cursor to the beginning of a line (specified by the line number).
MARK	Puts an invisible marker at the current position and associates it with the name you specify. Later, you can return to the marked position by using the GO TO command.
MOVE BY LINE	In forward direction: moves the cursor to the end of the current line or, if the cursor is already at the end of a line, to the end of the next line. In reverse direction: moves the cursor to the beginning of the current line or, if the cursor is already at the beginning of a line, to the beginning of the previous line. On VT200, VT300, and VT400 series terminals, EVE defines the F12 key as MOVE BY LINE. On VT100 series terminals EVE defines the Minus key on the keypad as MOVE BY LINE.
MOVE BY PAGE	Moves the cursor to the next or previous page break (form feed), depending on the current direction. If there is no page break in the current direction, the cursor moves to the bottom or top of the buffer.
MOVE BY WORD	In forward direction: moves the cursor to the beginning of the next word or, if the cursor is already at the end of a line, to the beginning of the next line. In reverse direction: moves the cursor to the beginning of the previous word or, if the cursor is already at the beginning of a line, to the end of the previous line.
NEXT SCREEN	Scrolls forward in the current buffer by the number of lines in the current window minus one. For example, if the current window is 12 lines long, the NEXT SCREEN command scrolls the cursor forward 11 lines. On VT200, VT300, and VT400 series terminals, EVE defines the E6 key (Next Screen) as NEXT SCREEN. On VT100 series terminals, EVE defines the KP0 key on the keypad as NEXT SCREEN.

(continued on next page)

Table 8–1 (Cont.) EVE Commands That Move the Cursor

Command	Function
NEXT WINDOW or OTHER WINDOW	Moves the cursor to the next window on your screen, if there is one. The cursor appears in the last location it occupied in that window. EVE defines GOLD Next Screen as NEXT WINDOW.
PREVIOUS SCREEN	Scrolls backward in the current buffer by the number of lines in the current window minus one. For example, if the current window is 12 lines long, the PREVIOUS SCREEN command scrolls the cursor backward 11 lines. On VT200, VT300, and VT400 series terminals, EVE defines the E5 key (Prev Screen) as PREVIOUS SCREEN. On VT100 series terminals, EVE defines the Period key on the keypad as PREVIOUS SCREEN.
PREVIOUS WINDOW	Moves the cursor to the previous window on your screen, if there is one. The cursor appears in the last location it occupied in that window. EVE defines GOLD Prev Screen as PREVIOUS WINDOW.
REVERSE	Sets the direction of the current buffer to reverse; that is, to the left and up. The direction of the buffer is shown in the status line.
SET CURSOR BOUND	Makes the cursor follow the flow of text. The cursor cannot move into an unused portion of the buffer. Similar to cursor behavior in EDT, WPS, and other editors.
SET CURSOR FREE	Default setting. You can move the cursor anywhere in the buffer and enter text there.
SET SCROLL MARGINS	Sets the top and bottom distances at which scrolling begins automatically as you move the cursor up and down. You specify these distances as numbers of lines or as a percentage of the window size. The default setting is 0; that is, scrolling starts when you move past the top or the bottom of the window.
SHIFT LEFT	Shifts the current EVE window to the left by the number of columns you specify. With SHIFT RIGHT and SHIFT LEFT commands, you can view the undisplayed portion of long lines of text without having to change the width of the window or use 132-column mode. The SHIFT LEFT command shifts the window only if you have used the SHIFT RIGHT command.
SHIFT RIGHT	Shifts the current EVE window to the right by the number of columns you specify. With SHIFT RIGHT and SHIFT LEFT commands, you can view the undisplayed portion of long lines of text without having to change the width of the window.
START OF LINE	Moves the cursor to the beginning of the current line. By default, EVE defines both Ctrl/H and GOLD ← as START OF LINE.

(continued on next page)

Table 8–1 (Cont.) EVE Commands That Move the Cursor

Command	Function
TOP	Moves the cursor to the beginning of the current buffer (upper left corner). By default, EVE defines GOLD ↑ as TOP.

8.7.4 Tutorial: Moving the Cursor in EVE

The following tutorial shows how to move the cursor through a buffer:

1. Invoke EVE and create the buffer SCHEDULE.DAT with the following command:

   ```
   $ EDIT/TPU SCHEDULE.DAT
   ```

   EVE puts the cursor at the top of the buffer and waits for you to enter text.

2. Enter the following text.

   ```
   Schedule for 1 July
   10:00 AM meeting with supervisor
   Read and review memo from Sally
   Work on Pascal program
   ```

   The [End of file] marker moves down in the buffer as you enter text and the cursor is positioned at the end of the text you inserted.

3. Enter the TOP command to move the cursor to the beginning of the file.
4. Press Ctrl/E to move the cursor to the end of the first line of text. Ctrl/E works the same way in EVE as it does in DCL.
5. Enter the BOTTOM command to move the cursor to the end of the buffer.
6. Press the up arrow key to move the cursor up one line to the fourth line of text.
7. Press the Change Direction key to change the current buffer direction to reverse.
8. Press the Move by Line key to move the cursor to the beginning of the third line of text.
9. Enter the command LINE 1 to move the cursor to the beginning of the first line of the buffer.
10. To exit from EVE, press Ctrl/Z.

8.8 Entering Text

8.8.1 Overview

You can enter keyboard characters, entire files, and special nonprinting characters (such as control characters) into the buffer that you are currently editing. You can use the keypad or commands to enter text. You can also add text, files, and special characters to the buffer.

8.8.2 Adding Text

You can type characters at the keyboard and add them to the buffer at the current cursor position. The characters you type either supplement or replace existing characters, depending on whether the buffer is in insert or overstrike mode.

8.8.3 Including Files

You can add an entire file by pressing the Do key and entering the EVE command INCLUDE FILE. Type the file specification at the File to include: prompt and press the Return key. Regardless of the current mode (insert or overstrike) of the buffer, EVE inserts the entire contents of the specified file into the buffer just before the line where the cursor currently appears.

You can use wildcards in the file specification. If there is more than one match for a file specification with a wildcard, EVE displays a list of choices and prompts you to provide a more complete file specification. If the specified file does not exist, EVE displays a message stating that it could not include the file.

8.8.4 Special Nonprinting Characters

You can use the QUOTE command to add special nonprinting characters by pressing Ctrl/V followed by the special character. For example, to insert an escape character into the buffer, press Ctrl/V followed by Ctrl/[. The special character either supplements or replaces existing characters, depending on whether the buffer is in insert or overstrike mode.

8.8.5 EVE Editing Keys for Entering Text

The following table shows the EVE editing keys that you can use to enter text:

Key or Key Sequence	Function
Ctrl/A	Same as the CHANGE MODE command. Changes the editing mode for the current buffer as shown in the highlighted status line. In insert mode, EVE inserts text at the character position, moving existing text to accommodate the insertion. In overstrike mode, EVE overwrites text at the current position. On VT200, VT300, and VT400 series terminals, EVE defines the F14 key as CHANGE MODE. On VT100 series terminals, EVE defines the Enter key on the keypad as CHANGE MODE.

Key or Key Sequence	Function
Ctrl/V	Same as the QUOTE command. You can insert nonprinting characters or control codes. To search for special characters, first press the Find key, then press Ctrl/V and the special character to be found. Activate the search by pressing the Return key.

8.8.6 EVE Commands for Entering Text

The following table shows the commands that you can use to enter text:

Command	Function
CHANGE MODE	Same as Ctrl/A. Changes the current editing mode as shown in the highlighted status line. In insert mode, EVE inserts text at the current position, moving existing text to accommodate the insertion. In overstrike mode, EVE overwrites text at the current position. On VT200, VT300, and VT400 series terminals, EVE defines the F14 key as CHANGE MODE. On VT100 series terminals, EVE defines the Enter key on the keypad as CHANGE MODE.
INCLUDE FILE	Inserts the contents of the specified file into the current buffer at the line above the cursor position. This is useful for combining files.
INSERT MODE	Sets the mode of the current buffer to insert, as opposed to overstrike. In insert mode, EVE inserts text at the current position, moving existing text to accommodate the insertion.
OVERSTRIKE MODE	Sets the mode of the current buffer to overstrike, as opposed to insert. In overstrike mode, EVE overwrites text at the current position.
QUOTE	Same as Ctrl/V. Enters a nonprinting character or a control code that you specify by pressing a key. You can quote a control code or other character when you enter a string for the FIND or REPLACE commands. For example, you can quote the Tab key to search for tab characters.

8.8.7 Setting Buffer Mode

Before you begin typing text, check whether your buffer is in insert mode or overstrike mode.

To determine the mode your buffer is in, look at the highlighted status line. If the buffer is in insert mode, text is inserted at the cursor position and text that already appears in the buffer moves to accommodate your insertions. If the buffer is in overstrike mode, text that you type at the keyboard is inserted at the cursor position and the text that already appears in the buffer is overwritten as the cursor moves through it.

To change from one mode to another, press Ctrl/A.

8.8.8 Tutorial: Adding Text

Use the following tutorial to add text to a file in both insert mode and overstrike mode:

1. Invoke EVE to edit the existing file SCHEDULE.DAT .
2. Check the highlighted status line to ensure that EVE is in insert mode.
3. If EVE is in overstrike mode, press Ctrl/A to change to insert mode.
4. Move the cursor to the first letter *s* in the word *supervisor*, type *Engineering*, and press the space bar.

   The word *Engineering* is inserted in your text buffer, and the rest of the text on the line moves to the right.

```
Schedule for 1 July
10:00 AM meeting with Engineering supervisor
Read and review memo from Sally
Work on Pascal program
[End of file]

Buffer: SCHEDULE.DAT                  | Write | Insert | Forward
```

5. Press Ctrl/A to change to overstrike mode.
6. Move the cursor to the letter *S* in the word *Sally* and type *Peggy*.

   The word *Peggy* is placed in the buffer, overwriting the word *Sally*.

```
Schedule for 1 July
10:00 AM meeting with Engineering supervisor
Read and review memo from Peggy
Work on Pascal program
[End of file]

Buffer: SCHEDULE.DAT                | Write | Overstrike | Forward
```

7. To exit from EVE, press Ctrl/Z.

8.9 Erasing and Restoring Text

8.9.1 Overview

With EVE, you can easily erase text or correct mistakes made during an editing session. If you erase text by mistake, you can restore the most recently erased text to its former location or, by moving the cursor, to another location.

To erase text from your buffer, move the cursor to the text you want to erase and press the appropriate editing key or enter the appropriate EVE command.

8.9.2 EVE Editing Keys for Erasing and Restoring Text

Table 8–2 shows EVE editing keys that erase and restore text.

Table 8–2 EVE Editing Keys for Erasing and Restoring Text

Key or Key Sequence	Function
Delete key or Delete	Erases the character to the left of the cursor. Same as the DELETE command. If pending delete is enabled, DELETE erases text in the select range and puts it into the Restore Selection buffer. For more information about using pending delete, see Section 8.10.
Ctrl/J	Same as ERASE WORD. Erases the current word or, if the cursor is between words, erases the next word. On VT200, VT300, and VT400 series terminals, EVE defines the F13 key as ERASE WORD. On VT100 series terminals, EVE defines the Comma key on the keypad as ERASE WORD.
Ctrl/U	Same as ERASE START OF LINE. Erases characters left of the cursor to the start of the line.
GOLD Insert Here	Same as RESTORE. Reinserts, at the current position, the word, line, or sentence that you just erased with an EVE command or editing key.
GOLD F13	Same as RESTORE WORD (except with the WPS keypad). Reinserts, at the current position, the word that you last erased.

8.9.3 EVE Commands for Erasing and Restoring Text

Table 8–3 shows EVE commands that erase and restore text.

Table 8–3 EVE Commands for Erasing and Restoring Text

Command	Function
DELETE	Erases the character to the left of the cursor. In insert mode, EVE moves existing text to accommodate the deleted character. In overstrike mode, EVE replaces the character with a space. At the start of a line, DELETE erases the carriage return for the previous line (regardless of mode) and the current line moves up. If pending delete is enabled, DELETE erases text in the select range and puts it into the Restore Selection buffer. For more information about using pending delete, see Section 8.10.
ERASE CHARACTER	Erases the character the cursor is on. In insert mode, EVE moves existing text to accommodate the deleted character. In overstrike mode, EVE replaces the character with a space. If the cursor is at the end of the line, the carriage return is erased—regardless of the mode—and the next line moves up.
ERASE LINE	Erases from the current character to the end of the line, appending the next line to the end of the current line. If the cursor is at the end of the line, only the carriage return is erased and the next line moves up.
ERASE PREVIOUS WORD	Erases the previous word or the word the cursor is on. If the cursor is between words or on the first character of a word, the previous word is erased. If the cursor is in the middle of a word, all of that word is erased (same as ERASE WORD). If the cursor is at the start of a line, the carriage return at the end of the previous line is erased and the current line moves up.
ERASE START OF LINE	Erases the current line of text, starting with the character left of the cursor until the start of the line. If you are already at the start of a line, nothing is erased.
ERASE WORD	Erases the current word or, if the cursor is between words, erases the next word. Same as Ctrl/J. On VT200, VT300, and VT400 series terminals, EVE defines the F13 key as ERASE WORD. On VT100 series terminals, EVE defines the Comma key on the keypad as ERASE WORD. If the cursor is at the end of the line, only the carriage return is erased and the next line moves up.
RESTORE	Reinserts, at the current position, the word, line, or sentence that you last erased with an EVE command or editing key. RESTORE does not restore single characters. EVE defines GOLD Insert Here as RESTORE.

(continued on next page)

Table 8–3 (Cont.) EVE Commands for Erasing and Restoring Text

Command	Function
RESTORE CHARACTER	Reinserts, at the current position, the character you last erased with an EVE command or editing key. In overstrike mode, the restored character replaces the character the cursor is on. In insert mode, the restored character is inserted at the cursor position and existing text moves to accommodate it.
RESTORE LINE	Reinserts, at the current position, the line that you last erased with an EVE command or editing key.
RESTORE SELECTION	Reinserts, at the current position, the text last erased with a pending delete operation. For more information about using pending delete, see Section 8.13.
RESTORE WORD	Reinserts, at the current position, the word that you last erased with an EVE command or editing key. EVE defines GOLD F13 as RESTORE WORD (except with the WPS keypad).

8.9.4 Tutorial: Erasing and Restoring Text

Use the following tutorial to erase and restore text:

1. Invoke EVE to create the buffer RHYMES.DAT and enter the following text:

   ```
   She rhymes with tree,
   also with bee,
   and this one makes three.
   ```

2. Move the cursor to the letter *l* in the word *also*. Enter the ERASE LINE command.

   EVE erases all characters from the letter *l* in *also* to the end of the line and appends the next line to the current line.

   ```
   She rhymes with tree,
   aand this one makes three.
   ```

3. Move the cursor to the letter *y* in the word *rhymes*. Enter the ERASE WORD command.

   EVE erases the word *rhymes* and shifts the remaining text to the left.

   ```
   She with tree,
   aand this one makes three.
   ```

4. Move the cursor to the second letter *a* on the second line. Enter the RESTORE LINE command.

   EVE restores the last line that was erased, in this case, *lso with bee,*.

   ```
   She with tree,
   also with bee,
   and this one makes three.
   ```

5. Move the cursor to the letter *w* in the word *with* on the first line. Enter the RESTORE WORD command.

EVE restores the last word that was erased, in this case, *rhymes*.

```
She rhymes with tree,
also with bee,
and this one makes three.
```

6. To exit from EVE, press Ctrl/Z.

Section 8.10 describes the functions of the SELECT and REMOVE commands, which can be used together to erase text from a buffer.

8.10 Moving Text

8.10.1 Overview

You can use EVE commands to select sections of text for copying, moving, deleting, or other editing operations. This section discusses how to move text.

For information on how to move text from one buffer to another, see Section 8.19.

You can also select a rectangular area (a box) of text rather than a linear range of text to move, erase, or duplicate text. For information about using box editing commands, see Section 8.12.

8.10.2 How to Move Text

To move text, follow these steps:

Step	Task
1	Once you have invoked a file in EVE, place the cursor on the first character you want to move.
2	Press the Select key.
3	Move the cursor to one character beyond the last character you want to move. (In reverse direction, move the cursor to the last character, not one beyond.) The text to be moved is highlighted in **reverse video**. (If you decide not to remove text from the buffer, press the Select key again to cancel the selection.)
4	Press the Remove key. EVE deletes the highlighted text from your screen and places it in the Insert Here buffer.
5	Press the Insert Here key to insert text. EVE inserts the text at the cursor location. You can insert the text contained in the Insert Here buffer any number of times at any cursor location until you select a new section of text and put that new text in the Insert Here buffer. The Insert Here buffer contains whatever text was last copied or removed.

8.10.3 EVE Editing Keys That Move Text

The following table describes EVE editing keys used to move text:

Key or Key Sequence	Function
Insert Here	Same as the INSERT HERE or PASTE command. Inserts, at the current position, text that you removed or copied.
Remove	Same as the REMOVE or CUT command. Removes the text that is marked with SELECT or highlighted by FIND and places it in the Insert Here buffer.
Select	Marks text (highlighting it in reverse video) from the initial cursor location to wherever you move the cursor. The text that is highlighted is called the select range. To cancel the selection, press the Select key again or use RESET.
GOLD Select	Same as RESET. Cancels any of the following and resets the direction of the buffer to forward: • Highlighting of a select or found range • A press of the GOLD key (or GOLD *n* combination for a repeat count) • An incomplete or recalled command line, or Choices buffer display • The output of SHOW, SHOW DEFAULTS BUFFER, SHOW SUMMARY, or SHOW WILDCARDS, thereby returning you to the buffer you were working in
GOLD Remove	Same as the STORE TEXT or COPY command. Copies text that is marked with SELECT or FIND, putting it in the Insert Here buffer. Text that is copied is not removed from its original position.

8.10.4 EVE Commands That Move Text

The following table describes EVE commands used to move text:

Command	Function
INSERT HERE or PASTE	Inserts the text you copied or removed. By default, EVE defines the E2 key (Insert Here on the minikeypad on VT200, VT300, and VT400 series terminals) and the KP9 key (on VT100 series terminals) as INSERT HERE.
REMOVE or CUT	Removes the text that was marked with SELECT or highlighted by FIND, and places it in the Insert Here buffer. By default, EVE defines the E3 key (Remove on the minikeypad on VT200, VT300, and VT400 series terminals) and the KP8 key (on VT100 series terminals) as REMOVE.
RESET	Cancels any of the following and resets the direction of the buffer to forward: • Highlighting of a select or found range • A press of the GOLD key (or GOLD *n* combination for a repeat count) • An incomplete or recalled command line, or Choices buffer display • The output of SHOW, SHOW DEFAULTS BUFFER, SHOW SUMMARY, or SHOW WILDCARDS, thereby returning you to the buffer you were working in
RESTORE SELECTION	Reinserts the text erased by a pending delete operation. For more information about using pending delete, see Section 8.13.
SELECT	Highlights text in reverse video from the initial cursor location to wherever you move the cursor. The text that is highlighted is called the select range. To cancel the selection, enter the SELECT command again or use RESET. By default, EVE defines the E4 key (Select on the minikeypad on VT200, VT300, and VT400 series terminals) and the KP7 key (on VT100 series terminals) as SELECT.
SELECT ALL	Highlights all text in reverse video in the current buffer regardless of the cursor position. The text that is highlighted is called the select range. To cancel the selection, enter the SELECT command or use RESET. The SELECT ALL command temporarily disables pending delete to avoid accidentally erasing all of the buffer.
SET NOPENDING DELETE	Default setting. Disables deletion of selected text when you use the Delete key or type new text. If you select text in the buffer, typing new text adds characters to the select range and using the Delete key erases only the character to the left of the cursor.

Command	Function
SET PENDING DELETE	Enables pending delete, which lets you quickly erase blocks of text. First enable pending delete, then use the SELECT command to choose the text you want to erase. Erase the text by pressing the Delete key (or any other key on the alpha-numeric keypad). To reinsert what you deleted, move the cursor to where you want the text and enter the RESTORE SELECTION command. The default is SET NOPENDING DELETE.
STORE TEXT or COPY	Copies text that was marked with SELECT or FIND, placing it in the Insert Here buffer. Text that is copied is not removed from its original position.

8.10.5 Tutorial: Moving Text

Use the following tutorial to select, remove, and insert text from one location to another:

1. Invoke EVE to edit the file RHYMES.DAT.
2. Move the cursor to the beginning of the second line of RHYMES.DAT and press the Select key.
3. Press the down arrow key once.

   The second line of text is highlighted.
4. Press the Remove key.

   The second line of text is removed from the current buffer.

   ```
   She rhymes with tree,
   and this one makes three.
   [End of file]
   ```
5. Press the Return key twice and then press the Insert Here key.

   The text in the Insert Here buffer is inserted at the current cursor location.

   ```
   She rhymes with tree,


   also with bee,
   and this one makes three.
   [End of file]
   ```
6. To exit from EVE, press Ctrl/Z.

8.11 Copying Text

8.11.1 Overview

With the COPY command, you can copy text elsewhere. The STORE TEXT command is the same as the COPY command. You can substitute the STORE TEXT command wherever the COPY command is used in the following example.

8.11.2 Tutorial: Copying Text

Use this tutorial to copy text when the buffer is set in a forward direction:

1. Invoke EVE to edit the file RHYMES.DAT.
2. Move the cursor to the first line of text.
3. Press the Select key.
4. Press Ctrl/E to move the cursor to the end of the first line.
5. Enter the COPY command. The Insert Here buffer now contains a copy of the selected text.
6. Move the cursor to the line above *also with bee,*.
7. Press the Insert Here key. Your buffer should now look as follows:

```
She rhymes with tree,
```

```
She rhymes with tree,
also with bee,
and this one makes three.
[End of file]
```

8. Move the cursor to the beginning of the first line of text. Use the Select key and then the Remove key to delete the first line of text.
9. To exit from EVE, press Ctrl/Z.

8.12 Box Editing

8.12.1 Overview

You can edit text that has rectangular areas, or boxes, as well as standard linear ranges. For example, you can select a box containing a list or columns in a table, and then cut and paste the box or perform some other editing operation on the box.

8.12.2 Selecting a Box of Text

To select a box of text, follow these steps:

1. Put the cursor where you want to start the selection—typically, where you want the upper left corner of the box.
2. Enter the BOX SELECT command.
3. Move the cursor to where you want the diagonally opposite corner of the box—typically, moving from upper left to lower right.

As you move the cursor, text that you cross is highlighted in bold video (a regular selection uses reverse video). The box is defined by diagonally opposite corners. If you move from upper left to lower right, the character that the cursor is on is *outside* the box, that is, the lower right corner of the box is left of the cursor.

You can then edit the box by using any of the editing commands that ordinarily work on a linear or a rectangular range. You need not redefine keys. See the *Extensible Versatile Editor Reference Manual* for further information.

You can use FIND SELECTED if the selection does not cross lines or OPEN SELECTED. You can also use pending delete.

If you are going to make several box edits—for example, in editing multicolumn tables and lists—use the SET BOX SELECT command. SET BOX SELECT redefines several commands and keys as the corresponding BOX commands and makes other editing operations work on boxes instead of linear ranges.

To cancel a box selection, repeat SELECT or BOX SELECT, or use RESET.

8.12.3 Cutting and Pasting a Box of Text

Cutting a box usually pads the area with spaces to keep the column alignment of text to the right of the box. Pasting a box usually overwrites existing text. Tab characters in the box, or that overlap the box, are converted to spaces to keep the column alignment of text.

8.12.4 EVE Commands for Box Editing

The following table lists the EVE commands for box editing:

Command	Function
BOX COPY	Copies a box of text without removing it, so you can paste it elsewhere.
BOX CUT	Cuts a box of text so you can paste it elsewhere, usually padding the area with spaces to keep the column alignment of text to the right of the box.
BOX CUT INSERT	Cuts a box, making text to the right of the box ‘‘collapse’’ to the left, closing the gap.
BOX CUT OVERSTRIKE	Cuts a box, padding the area with spaces to keep the column alignment of text to the right of the box.
BOX PASTE	Pastes a box of text you copied or cut, usually overwriting existing text.
BOX PASTE INSERT	Pastes a box, pushing existing text to the right.
BOX PASTE OVERSTRIKE	Pastes a box, overwriting existing text.
BOX SELECT	Selects a box of text. Typically, you start at the upper left corner of the box and move the cursor to where you want the lower right corner.
RESTORE BOX SELECTION	Puts back (undeletes) a box erased with pending delete, usually overwriting existing text.
SET BOX NOPAD	Disables padding and overstriking for box editing unless the buffer is in overstrike mode.
SET BOX NOSELECT	Default setting. Disables box selection, cutting, and pasting. Commands such as SELECT, COPY, and REMOVE use standard linear ranges. To edit boxes, use BOX commands.

Command	Function
SET BOX PAD	Default setting. Enables automatic padding and overstriking for box editing, regardless of the buffer mode.
SET BOX SELECT	Enables box selection, making commands such as SELECT, REMOVE, and INSERT HERE the same as the corresponding BOX commands, without having to redefine keys.

8.12.5 Tutorial: Cutting and Pasting Text

Use this tutorial to select and then cut and paste a box of text:

1. Invoke EVE to create the buffer CITIES.DAT and enter the following text:

```
Rome    Paris   New York
London  Tunis   Boston
Tokyo   Bonn    Lisbon
```

2. Move the cursor to the left of the letter *P* in the word *Paris*. Enter the BOX SELECT command.
3. Move the cursor two spaces to the right of the second letter *n* in the word *Bonn*—the diagonally opposite corner of the box. The text that you cross is highlighted in bold video. Enter the BOX CUT command.

   EVE removes the box of text.
4. Move the cursor to the right of the column that begins with the words *New York*.
5. Enter the BOX PASTE command.

   EVE pastes the box of text into a new column, as follows:

```
    Rome        New York     Paris
    London      Boston       Tunis
    Tokyo       Lisbon       Bonn
[End of file]
```

8.12.6 SET BOX SELECT Commands

The following lists the SET BOX SELECT commands:

Command	Effect with SET BOX SELECT
INSERT HERE or PASTE	BOX PASTE
REMOVE or CUT	BOX CUT
RESTORE SELECTION	RESTORE BOX SELECTION
SELECT	BOX SELECT
STORE TEXT or COPY	BOX COPY

You can then select, cut, and paste a box by using the Select, Remove, and Insert Here keys, without having to redefine the keys.

8.13 Using Pending Delete

8.13.1 Overview

You can use pending delete to erase selected text. Pending delete refers to erasing a selection by typing new text, pressing the space bar, or by using delete (typically, pressing the Delete key).

With a box selection, pending delete works like BOX CUT, usually padding the area with spaces to keep the column alignment of text to the right of the box.

Pending delete gives you an alternative way of cutting and pasting text because pending delete does not use the Insert Here buffer. For more information about pending delete, see the EVE online help topic called Pending Delete.

8.13.2 Erasing a Selection with Pending Delete

To erase a selection using pending delete, follow these steps:

1. Invoke a file in EVE.
2. To enable pending delete, use the SET PENDING DELETE command. The default setting is SET NOPENDING DELETE.
3. Select the text you want to erase. You can use SELECT or BOX SELECT. (You cannot use SELECT ALL.)
4. Type new text or use the DELETE command.

8.13.3 Restoring a Selection That Was Erased with Pending Delete

To put back (restore) the text you erased with pending delete, follow these steps:

1. Put the cursor where you want to restore the text. If restoring a box selection, put the cursor where you want the upper left corner of the box to be.
2. Use RESTORE SELECTION. If a box selection was erased with pending delete, use RESTORE BOX SELECTION. If you used SET BOX SELECT, you can use RESTORE SELECTION (without having to redefine a key).

8.13.4 Effects on Box Editing

Restoring a box works like BOX PASTE, usually overwriting existing text. When using the SET BOX NOPAD command, the effects of box editing depend on the mode that the buffer is in (insert or overstrike, as shown in the status line):

- In insert mode, cutting a box makes text to the right of the box ‘‘collapse’’ to the left, closing the gap. Tab characters to the right of the box are also converted to spaces to keep the column alignment as the text collapses to the left. This method is useful for removing columns from a table or list, such as in turning a 4-column table into a 2-column table. Pasting a box pushes existing text to the right, which is useful for adding columns in the middle of a table.

- In overstrike mode, cutting a box pads the area with spaces to keep the column alignment of text to the right of the box. Pasting a box overwrites existing text. The effects are the same as with SET BOX PAD, which is the default setting.

The buffer mode also affects erasing a box with pending delete and restoring an erased box.

8.14 Finding and Replacing Text

8.14.1 Overview

With EVE commands, you can search for specific text in a buffer. You can search for every occurrence of specific text, and you can search for text that is on a single line or spans a line break. Also, you can search for text using wildcards. This section describes methods for searching and replacing text.

8.14.2 EVE Commands for Locating Text in a Buffer

Table 8–4 describes the EVE commands that locate text in a buffer.

Table 8–4 EVE Commands for Locating Text in a Buffer

Command	Function
FIND	Searches the current buffer for the text string you specify and highlights the found text. The text that is highlighted is called the found range.
FIND NEXT	Searches for the string of text you last specified with the FIND, REPLACE, or WILDCARD FIND command.
FIND SELECTED	Searches for a string of text you have selected, rather than for a typed string. The selection cannot cross more than one line.
SET FIND CASE EXACT	Enables case-exact searches. This is particularly useful to find or replace search strings in lowercase letters only.
SET FIND CASE NOEXACT	Default setting. Disables case-exact searches so that EVE finds any occurrence if you enter a search string in all lowercase letters.
SET FIND NOWHITESPACE	Default setting. Sets FIND and WILDCARD FIND commands to match tabs and spaces exactly as you specify in the search string and to search for strings that are entirely on one line.
SET FIND WHITESPACE	Sets FIND and WILDCARD FIND commands to treat spaces, tabs, and up to one line break as ‘‘white space’’ so you can search for strings of two or more words regardless of how they are separated.
SET WILDCARD VMS	Default setting on OpenVMS. Enables OpenVMS patterns for WILDCARD FIND.

(continued on next page)

Table 8–4 (Cont.) EVE Commands for Locating Text in a Buffer

Command	Function
SHOW WILDCARDS	Lists the wildcard patterns you can use with WILDCARD FIND.
WILDCARD FIND	Searches for a pattern of text, using wildcards.

8.14.3 Finding Text

Use the FIND command to locate specific text in the current buffer. By default, EVE defines the E1 key (Find key on VT200, VT300, and VT400 series terminals and the PF1 key on VT100 series terminals) as the FIND command.

If the search string contains all lowercase letters, EVE disregards the case of letters and locates any occurrence of the string. Thus, the search string *the* matches *the*, *THE*, *THe*, and *thE*. If the search string contains one or more uppercase letters, EVE finds only the occurrences of the string in which the case of each letter is exactly the same. Therefore, the only match for the search string *tHis* is *tHis*. For example:

1. Enter the FIND command.
2. Type the text (called the search string) that you want to locate.

8.14.4 Search Direction

The current direction of the buffer determines whether EVE first searches in a forward or reverse direction.

If EVE cannot find the string in the current direction but finds it in the opposite direction, EVE prompts you to change direction.

To search in the opposite direction, type YES (Y) and press the Return key. EVE moves the cursor to the first occurrence of the string in the opposite direction. The current direction in the highlighted status line does not change, however.

8.14.5 When a Search String is Found

When EVE finds the search string, the editor highlights it and moves the cursor to the first letter of the string. See the *Extensible Versatile Editor Reference Manual* for a listing of the editing commands you can use on a highlighted search string.

To cancel the highlighting, move the cursor off the search string or use the RESET command.

To find the next occurrence of the search string, press the Find key twice or enter the FIND NEXT command.

8.14.6 Setting Case-Exact Searches

If you want to match the case of your search exactly when searching for lowercase occurrences of a string, enter the SET FIND CASE EXACT command. Then when you enter a search string in all lowercase letters, EVE searches only for lowercase occurrences, skipping occurrences that contain uppercase letters.

The setting applies to the FIND, REPLACE, and WILDCARD FIND commands. You can save the setting in your section file or command file for future editing sessions. The default setting is SET FIND CASE NOEXACT.

EVE is sensitive to diacritical (accent) marks and locates only those occurrences of the string in which the diacritical marks are exactly the same. For example, in searching for *ë*, EVE does not find occurrences of *e*, *é*, *è*, or *ê*.

8.14.7 Example

In the following example, the commands enable case-exact searching and then find *digital* when it appears in lowercase only, skipping occurrences such as *Digital* or *DIGITAL*:

```
Command: SET FIND CASE EXACT
Command: FIND digital
```

8.14.8 Tutorial: Finding Text

Use this tutorial to use the FIND command with the existing file RHYMES.DAT:

1. Invoke EVE to edit RHYMES.DAT. The cursor appears on the first letter of the first line of the buffer, and the current direction is forward.
2. Press the Find key, type the letters *ree*, and press the Return key. The cursor moves to the letter *r* in the word *tree* and highlights the letters *ree*.
3. Press the Find key twice to find the next occurrence of the string *ree*. The cursor moves to the letter *r* in the word *three* and highlights the letters *ree*.

   When a search string is found and highlighted, you can use any command that works on a selected or found range except SPELL. Also, you cannot use a pending delete operation on a found range.
4. Enter the UPPERCASE WORD command.

   The UPPERCASE WORD command changes the case of the highlighted letters from lowercase to uppercase, as shown in the following example:

   ```
   She rhymes with tree,
   also with bee,
   and this one makes thREE.
   [End of file]
   ```

8.14.9 Tutorial: Using the FIND SELECTED Command

Use this tutorial to use FIND SELECTED to search for a string that is particularly complicated or is easily misspelled or mistyped:

1. Copy the text (from the previous tutorial) so that it is displayed twice in the buffer.
2. Move the cursor to the beginning of the string *rhymes with tree,* on the first line.
3. Enter the SELECT command.
4. Move the cursor to highlight the string and select text. Note that the selection cannot span more than one line.
5. Enter the command FIND SELECTED.

   The cursor moves to the next occurrence of the string *rhymes with tree,*. The selection is canceled and the found string appears in bold video.

8.14.10 Using Wildcards

You can use wildcards to search for text. The SHOW WILDCARDS command displays wildcard patterns for the current wildcard setting.

8.14.11 Tutorial: Using Wildcards

Use this tutorial to learn how to use wildcards:

1. Position the cursor at the beginning of the buffer.
2. Enter the command WILDCARD FIND **ee* to search for text strings ending in *ee*.

   ```
   She rhymes with tree,
   also with bee,
   and this one makes thREE.
   [End of file]
   ```

   EVE puts the cursor at the beginning of the line containing the *r* in *tree*.

8.14.12 Including White Space in a Search

Use the SET FIND WHITESPACE and SET FIND NOWHITESPACE commands to specify how the WILDCARD FIND and FIND commands treat the blank spaces between words, such as spaces, tabs, and line breaks.

The SET FIND NOWHITESPACE command enables the commands to search for multiword strings on a single line, matching spaces and tabs exactly as they are found. SET FIND NOWHITESPACE is the default search behavior.

The SET FIND WHITESPACE command enables the WILDCARD FIND and FIND commands to search for a string of two or more words regardless of how they are separated. It enables the FIND commands to search for a string that contains a single line break and more than one space or tab between words.

8.14.13 Marking Locations in Text

The MARK and GO TO commands are useful for editing a large file and then returning to a specific location later in the editing session. The following table describes the MARK and GO TO commands:

Command	Function
MARK	Puts an invisible mark at the current cursor position. The mark exists for the rest of an editing session or until you change it; it is not saved when you exit.
GO TO	Returns the cursor to the location labeled by the MARK command. If the labeled location is found in another buffer, EVE moves the cursor to the other buffer and puts that buffer into the current window.

To mark your position, enter the MARK command followed by a label name of your choice. The label name can be one or more printable characters, including alphanumeric and punctuation characters, spaces, and tab characters. To return the cursor to the marked location, enter the GO TO command followed by the label name.

8.14.14 Replacing Text

With the REPLACE command, you can replace a text string in the current buffer with another text string. This is useful if you have spelled a word incorrectly throughout a long file and you want to fix every occurrence of the misspelled word.

8.14.15 REPLACE Command and Case Sensitivity

The REPLACE command is case sensitive. If the old string has any uppercase letters, EVE searches for exact case matches. If the old string is all lowercase, EVE searches for any occurrence of the string regardless of its case. If the new string has any uppercase letters, EVE replaces the string exactly. If the old and new strings are all lowercase, EVE replaces the string according to the following rules:

- A capitalized version of the old string (first letter uppercase, others lowercase) is replaced by a capitalized version of the new string.
- An all-uppercase version of the old string is replaced by an all-uppercase version of the new string (otherwise, the old string is replaced by an all-lowercase version of the new string).

8.14.16 Case Handling by EVE

The following table shows how EVE uses the case of the strings:

Old String	New String	Highlight	Replacement
butter	margarine	butter	margarine
		Butter	Margarine
		BUTTER	MARGARINE
		BUtteR	margarine
Butter	margarine	Butter	margarine
butter	Margarine	butter	Margarine
		Butter	Margarine
		BUTTER	Margarine
		BUtteR	Margarine
Butter	Margarine	Butter	Margarine

8.14.17 SET FIND CASE EXACT Command

If you want to find or replace only lowercase occurrences of a string, enter the SET FIND CASE EXACT command. Then if you enter a search string in all lowercase, EVE searches for only lowercase occurrences, skipping occurrences that contain uppercase letters. The setting applies to FIND, REPLACE, and WILDCASE FIND commands.

The following table shows how EVE searches for and replaces only lowercase strings when you enter the SET FIND CASE EXACT command:

Old String	New String	Highlight	Replacement
butter	margarine	butter	margarine

The default case setting is SET FIND CASE NOEXACT.

8.14.18 REPLACE Command Responses

The following table shows the responses and their effect to the REPLACE command query:

Response	Effect
Yes	Replace this occurrence and find the next one. This is the default response. Press the Return key.
No	Skip this occurrence and find the next one.

Response	Effect
All	Replace all occurrences (no further prompting unless EVE finds an occurrence in the opposite direction).
Last	Replace this occurrence and stop here.
Quit	Skip this occurrence and stop here.

8.15 Using Command Line Qualifiers

8.15.1 Overview

When you invoke EVE, you can use command line qualifiers to specify advanced EVE editing features. When using the character-cell screen updater, the default insert or overstrike mode is determined by your terminal setting.

8.15.2 EDIT/TPU Command Line Qualifiers

The following table lists the qualifiers that you can use with the EDIT/TPU command to invoke EVE:

Qualifier	Default
Command file	/COMMAND=TPU$COMMAND.TPU
File creation	/CREATE
Debugging package	/NODEBUG
Specifying display mode	/DISPLAY=CHARACTER_CELL
Initialization file	/INITIALIZATION=EVE$INIT.EVE
Journaling	/JOURNAL
Modifying main buffer	/MODIFY
Specifying output	/OUTPUT=output-file
Read-only access	/NOREAD_ONLY
Recovery	/NORECOVER
Section files	/SECTION=TPU$SECTION
Start position	/START_POSITION=(1,1)
Work file	/WORK=SYS$SCRATCH:TPU$WORK.TPU$WORK

8.15.3 Starting in an Alternate Position

Start position qualifiers determine the row and column where the cursor first appears in the buffer that you specified on the command line.

For EVE, the default start position is 1,1—row 1, column 1, which is the upper left corner of the buffer. Use of start position qualifiers does not affect the initial cursor position when you

create another buffer during the editing session and does not limit the buffer size.

8.15.4 /START_POSITION Qualifier Format

The format of the start position qualifier is as follows:

/START_POSITION=(row[,column]

The fields are as follows:

/START_ POSITION	You must use the /START_POSITION= qualifier to the EDIT/TPU command.
row	The row that you want the cursor to be at when you invoke EVE.
column	The column that you want the cursor to be at when you invoke EVE.

8.15.5 Using the /START_POSITION Qualifier

Use the start position qualifier to begin editing at a particular line (or row) or at a particular character position (or column). For example, when you want to skip over a standard heading in a file or if a batch log file or error message tells you there is an error on a given line of a program, you can specify that line number as the starting row so that when you edit the program source file, the cursor moves directly to that line. The following command edits a file named test.com and puts the cursor on line 10, column 5:

```
$ EDIT/TPU TEST.COM /START_POSITION=(10,5)
```

If you want to start at a particular line in a file, you can omit the second parameter (the column).

8.15.6 Work Files

Work file qualifiers determine the work file that is used to swap memory for editing very large files. There is one work file per editing session. The work file is a temporary file that is automatically deleted when you exit.

The default work file is named TPU$WORK.TPU$WORK. EVE creates the work file in SYS$SCRATCH unless you specify otherwise.

8.15.7 Specifying Work Files

There are two ways to specify a different work file:

- Define the logical name TPU$WORK.

  This is useful if you want the work file to be created in an area other than SYS$SCRATCH, such as on a larger disk. You can put the definition in your LOGIN.COM file.

- Use the /WORK= qualifier and specify the work file.

  This overrides any definition of the TPU$WORK logical name. For example, the following command invokes EVE and specifies the work file to be SYS$SCRATCH:MYWORK.TPU$WORK:

```
$ EDIT/TPU /WORK=MYWORK
```

If you want the work file to be created in an area other than SYS$SCRATCH, use a complete file specification, including the device (disk) and directory. You cannot use wildcards to specify the work file.

8.15.8 Modifying the Main Buffer

Modifying qualifiers determine whether you can modify the buffers specified on the command line. Modifications do not affect other buffers you create during the editing session.

By default, you can modify the buffer by editing text in it. When you exit, EVE writes out the buffer to a file if the buffer has been modified.

Use /NOMODIFY to examine a file without making any changes. You can then use cursor-movement commands but you cannot change the text.

If you specify neither /MODIFY nor /NOMODIFY, your application determines if you can modify the buffer. EVE's default behavior is to modify the buffer.

8.15.9 Overriding the /READ_ONLY Qualifier

Use /MODIFY to override the effect of /READ_ONLY or /NOWRITE. Use /MODIFY with /READ_ONLY or /NOWRITE to practice editing operations without writing a file on exiting. For example, the following command invokes EVE, making the buffer you specified on the command line read-only (or no-write) and making it modifiable:

```
$ EDIT/TPU /READ_ONLY /MODIFY
```

In EVE, you can set or change the modification attribute of the buffer by using SET BUFFER commands.

8.16 Alternate Methods to Invoke EVE

8.16.1 Overview

You can invoke EVE using four different methods: from search lists, with wildcards, with wildcard directory names, or with multiple input files.

8.16.2 Example: Invoking EVE from a Search List

In the following example, if the first file in the search list exists, EVE copies that file (HIRING.DAT) into a buffer and uses the file name and file type as the buffer name. If the file does not exist, EVE tries to get the second file (PROMOTION.LIS), and so on. If none of the files in the search list exist, EVE creates an empty buffer named HIRING.DAT because that is the first file in the search list.

```
$ DEFINE STAFFMEMOS HIRING.DAT,PROMOTION.LIS,SALARY.TXT
$ EDIT/TPU STAFFMEMOS
```

8.16.3 Invoking EVE with Wildcards

When you invoke EVE to edit an existing file, you can use the asterisk (*) wildcard character as a substitute for some or all of the characters in the file name and file type. To use wildcards in EVE, follow the same rules as using wildcards in DCL. You can use the percent sign (%) wildcard character as a substitute for a single character at a time, and you can use the ellipsis ([...]) wildcard character as a substitute for a directory specification. If only one match is made, the file is displayed on your screen. If more than one match is made, EVE displays a list of matching files and prompts you to provide a more complete file specification. If no match is made, EVE creates a buffer named Main.

If more than one file matches your wildcard request, EVE displays the matching files so you can choose the one you want.

If no matching file is found, EVE creates an empty buffer named Main. If you use a search list or wildcard directory to specify an input file, EVE gets the first matching file found without displaying the $CHOICES$ buffer. For information about using the $CHOICES$ buffer, see the EVE online help topic called Choices Buffer.

8.16.4 Examples

- In the following example, a list of all files with the file type .TXT will be displayed:

```
$ EDIT/TPU *.TXT
```

- If you specify *.TXT, two files (LETTER.TXT and MEMO.TXT) match your wildcard request. In this case, EVE gives you a list in a second window in a system buffer named $CHOICES$.

8.16.5 Invoking EVE with Wildcard Directory Names

You can use wildcards in a directory name ([...]) to invoke EVE and work either in your current directory or in a subdirectory of the current directory.

This way of handling a search list or wildcard directory applies not only to the EDIT/TPU command, but also to EVE commands that use a file specification as a parameter. The following EVE commands use a file specification as a parameter:

@ (at sign)
GET FILE
INCLUDE FILE
OPEN
OPEN SELECTED
RECOVER BUFFER

8.16.6 Example

In the following example, EVE searches through the directory tree and gets the first PINK.TXT file found, if there is one.

```
$ EDIT/TPU [...]PINK.TXT
```

8.16.7 Invoking EVE with Multiple Input Files

You can specify multiple input files on the command line that invokes EVE. The file names must be separated by commas with optional white space. If wildcard characters are present in the file names, EVE displays the matching files only for the first wildcard file name that has more than one match. For the other ambiguous file names, EVE outputs a warning message.

8.17 Journaling and Recovery

8.17.1 Overview

Journal files record your edits so that if a system failure interrupts your editing session, you can recover your work.

Buffer-change journaling creates a separate journal file for each text buffer you create. This is the EVE default. Buffer-change journaling works both on DECwindows and on character-cell terminals. You recover one buffer at a time, typically by using RECOVER BUFFER commands in EVE. You can recover buffers from different editing sessions. The recovery restores only your text—it does *not* restore settings, key definitions, or the contents of system buffers (such as the Insert Here buffer) before the system failure.

8.17.2 Disabling Journaling

You can disable journaling when you invoke EVE by using the /NOJOURNAL qualifier on your command line. This is useful when you use EVE to examine a file without making any edits or for demonstration sessions.

8.17.3 Running File Backups

EVE file backups are disabled and cannot be enabled because the OpenVMS file system provides version numbers; therefore, no EVE mechanism is needed.

8.17.4 Using Buffer-Change Journaling

Buffer-change journaling creates a journal file for each text buffer. (EVE does not create buffer-change journal files for system buffers such as the Insert Here buffer, DCL buffer, or $RESTORE$ buffer.) As you edit a buffer, the journal file records the changes you make, such as erasing, inserting, or reformatting text. When you exit from EVE or when you delete the buffer, the journal files are deleted. If a system failure interrupts your editing session, the journal files are saved. Your last few keystrokes before the system failure may be lost.

8.17.5 EVE Commands for Buffer Change Journaling and Recovery

The following table summarizes the EVE commands for buffer-change journaling and recovery:

Command	Function or Effect
RECOVER BUFFER	Recovers a specified buffer by using the journal file for the buffer. You can specify the name of the buffer or file you want to recover or the name of the journal file for the buffer.
RECOVER BUFFER ALL	Recovers all your text buffers, one at a time, by using the journal files for the buffers, if there are any.
SET JOURNALING	Enables buffer-change journaling for a buffer that you specify.
SET JOURNALING ALL	Enables buffer-change journaling for all your buffers. This is the default setting.
SET NOJOURNALING	Disables buffer-change journaling for a buffer that you specify.
SET NOJOURNALING ALL	Disables buffer-change journaling for all your buffers.

8.17.6 Buffer Change Journal Files

Buffer-change journal files are written in a directory defined by the logical name TPU$JOURNAL. By default, this directory is SYS$SCRATCH, which is typically your top-level (login) directory. You can redefine the TPU$JOURNAL logical name to have the journal files written to a different directory. For example, the following commands create a subdirectory called [USER.JOURNAL] and then define TPU$JOURNAL as this subdirectory:

```
$ CREATE/DIRECTORY [USER.JOURNAL]
$ DEFINE TPU$JOURNAL [USER.JOURNAL]
```

You can put the definition in your LOGIN.COM file.

Buffer-change journal files may be quite large (even larger than the text files you edit). Because of the potential size of buffer-change journal files and because there is a journal file for each text buffer, you may want to define TPU$JOURNAL as a directory or subdirectory on a large disk, rather than as SYS$SCRATCH.

8.17.7 Deriving Buffer Change Journal Names

Buffer-change journal file names are derived from the name of the file or buffer being edited and the default file type for the operating system. To find out the name of the journal file for the current buffer, enter the SHOW command at the EVE prompt. The SHOW command displays the name of your input file, output file, your journal file, and other information about your current buffer.

8.17.8 Buffer Change Journal File Names

Table 8–5 shows the buffer-change journal file names:

Table 8–5 Buffer-Change Journal File Names

Text Buffer Name	Buffer-Change Journal File
JABBER.TXT	JABBER_TXT.TPU$JOURNAL
GUMBO_ RECIPE.RNO	GUMBO_RECIPE_RNO.TPU$JOURNAL
MAIN	MAIN.TPU$JOURNAL
LATEST NEWS	LATEST_NEWS.TPU$JOURNAL

8.17.9 Using Buffer-Change Journaling to Recover Edits

There are two ways to recover your edits with buffer-change journal files:

- Use the /RECOVER qualifier on the EDIT/TPU command line when you invoke EVE.
- Use RECOVER BUFFER commands within EVE.

8.17.10 Using the /RECOVER Qualifier

In the following example, you are editing a file named JABBER.TXT when a system failure interrupts your editing session. You then use system recovery to recover your edits.

```
$ EDIT/TPU JABBER.TXT
                .
                .
                .
   ***  system failure  ***
                .
                .
                .
$ EDIT/TPU JABBER.TXT/RECOVER
```

8.17.11 Using the RECOVER BUFFER Command

To use the recover buffer command, follow this procedure:

Step	Task
1	Invoke EVE and enter the following command to recover your text: `Command: RECOVER BUFFER` *`filename.txt`* If the buffer-change journal file is available, EVE shows the following information and asks if you want to recover that buffer: Name of the buffer Original input file for the buffer, if any Output file for the buffer, if any Source file for recovery, if any Starting date and time of the editing session Journal file creation date and time

Step	Task
2	Press the Return key to recover your buffer. If you do not want to recover your buffer, type No and press the Return key. If you delete or rename the source file for recovery, the recovery fails. The source file is either the file initially read into the buffer (if any) or the last file written before the system failure. If the buffer you want to recover exists (usually the Main buffer), EVE first deletes that buffer and then does the recovery. If the buffer you want to recover has been modified, EVE asks you whether to delete the buffer before recovering.

8.17.12 How to Recover When You Are Unsure of the File Name

If you are unsure of the buffer names or journal file names, specify the asterisk (*) wildcard, as follows:

```
Command: RECOVER BUFFER *
```

EVE then displays a list of all your available journal files so you can choose the one you want. The list appears in an EVE system buffer named $CHOICES$ in a second window. For information about using the $CHOICES$ buffer, see the EVE online help topic called Choices Buffer.

8.17.13 How to Recover All Buffers

To recover all your text buffers—one at a time—use the RECOVER BUFFER ALL command. EVE then tries to recover each text buffer for which there is a buffer-change journal available. The effect is the same as repeating the RECOVER BUFFER command without having to specify buffer names or journal file names. For each text buffer, EVE displays information such as the buffer name, the files associated with the buffer, and the time and date the journal file was created. EVE prompts you for one of the following:

Response	Effect
Yes	Recovers the buffer and then asks you whether to recover the next buffer, if there is one. This is the default response. Press the Return key.
No	Skips this recovery. If there is another buffer to recover, EVE asks you about the other buffer.
Quit	Cancels—does not recover the buffer and does not continue recovery operations.

8.17.14 Disabling Buffer-Change Journaling

You can disable buffer-change journaling for a particular buffer by using the SET NOJOURNALING command. To disable buffer-change journaling for all your buffers, use the SET NOJOURNALING ALL command.

8.17.15 Enabling Buffer-Change Journaling

If you disabled buffer-change journaling, you can enable journaling by using the SET JOURNALING command. The following command enables journaling for a buffer named JABBER.TXT:

```
Command: SET JOURNALING JABBER.TXT
```

If you invoked EVE without journaling and then want to enable buffer-change journaling during the editing session, use the SET JOURNALING ALL command (which is the EVE default).

You cannot enable buffer-change journaling if the buffer has been modified. In such a case, EVE displays the following message:

```
Command: SET JOURNALING MEMO.TXT
Buffer MEMO.TXT is not safe for journaling
```

You should first write out (save) the buffer by using the WRITE FILE or SAVE FILE command and then enable journaling.

8.18 EVE Formatting Commands

8.18.1 Overview

EVE provides commands that let you format your text by setting margins, tabs, and word wrap. You can center lines, take extra white space out of text, and insert page breaks.

8.18.2 EVE Editing Keys and Their Functions

The following table shows EVE editing keys and describes their functions:

Key or Key Sequence	Function
Return or Ctrl/M	Inserts a carriage return at the current position either to start a new line of text or to terminate a command you are typing. On VT200, VT300, and VT400 series terminals, EVE also defines the Enter key as Return.
Tab or Ctrl/I	Inserts a tab character at the current position according to the tab modes and at the tab stops currently set.
Ctrl/L	Inserts a form-feed character at the current position to mark the beginning of a new page. A page break appears as a small double F (F_F) and is always on a line by itself. Same as INSERT PAGE BREAK.

8.18.3 EVE Text Formatting Commands and Their Functions

The following table shows EVE text formatting commands and describes their functions:

Command	Function
CAPITALIZE WORD	Changes the case of a word, making the first letter uppercase and the rest of the letters lowercase. Works on a range, box, or single word.
CENTER LINE	Centers the current line between the left and right margins. The cursor moves with the line, remaining on the same character as the line moves.
CONVERT TABS	Converts tab characters to the appropriate number of spaces in a box, a range, or the entire buffer.
FILL	Reformats the current paragraph, range, or box according to the margins of the buffer, so the maximum number of words fits on a line. When you fill a select range or found range, the FILL or FILL RANGE command does not reformat a line that begins with a page break, a DIGITAL Standard Runoff (DSR) command, or DOCUMENT tag; it does reformat the other lines in the range. Filling a range does not delete blank lines. For more information about select range, see Section 8.10.
FILL PARAGRAPH	Reformats the paragraph that the cursor is in according to the margins set for the buffer. When you fill a paragraph, the FILL command does not reformat a line that begins with a page break, DSR command, or DOCUMENT tag; it does reformat the other lines in the paragraph.
FILL RANGE	Reformats the range or box according to the current margin settings. When you fill a select range or found range, the FILL or FILL RANGE command does not reformat a line that begins with a page break, DSR command, or DOCUMENT tag; it does reformat the other lines in the range. Filling a range does not delete blank lines.
INSERT PAGE BREAK	Inserts a form-feed character at the current position to mark the beginning of a new page. A page break appears as a small double F (F_F) and is always on a line by itself. By default, Ctrl/L is defined as INSERT PAGE BREAK.
LOWERCASE WORD	Changes the current word, range, or box to lowercase.

Command	Function
PAGINATE	Inserts a ''soft'' page break for a 54-line page. A soft page break appears as a form feed followed by the null character (FF NUL). When you enter the PAGINATE command, EVE moves back to the previous page break (if any) then checks ahead for page breaks within the next 54 lines. If any soft breaks are found within those 54 lines, EVE removes them. EVE then moves down 54 lines, inserts a soft break, and puts the cursor on the next line. The soft break is inserted on a line by itself. If a hard page break (form feed only) is found within the 54 lines, EVE stops on the line after the hard break, in case you want to erase the break.
SET LEFT MARGIN	Sets the left margin in the current buffer. The left margin must be greater than 0 but less than the right margin. By default, the left margin is 1 (leftmost column).
SET RIGHT MARGIN	Sets the right margin for the current buffer. The right margin must be greater than the left margin. By default, the right margin is one less than the width. The width is typically 80, so the default margin is typically 79.
SET PARAGRAPH INDENT	Specifies the number of spaces to be added to or subtracted from the first line of paragraphs you create or reformat. The default is 0 (no indent).
SET TABS AT	Sets tab stops at the columns that you specify. The column numbers must be in ascending order and separated by spaces. By default, tab stops are set every eight columns. The command does not affect the hardware tab settings of your terminal.
SET TABS EVERY	Sets tab stops at the specified interval. By default, tab stops are set every eight columns. The command does not affect the hardware tab settings of your terminal.
SET TABS INSERT	Default setting. Changes the tab mode so that EVE inserts a tab character at the current column when you press the Tab key. The cursor and text move to the next tab stop.
SET TABS MOVEMENT	Changes the tab mode so the Tab key becomes a cursor-movement key. Pressing the Tab key moves the cursor to the next tab stop but does not insert a tab character.
SET TABS SPACES	Changes the tab mode to insert an appropriate number of spaces, rather than a tab character, when the Tab key is pressed. Previously existing tab characters are not affected.
SET TABS INVISIBLE	Default setting. Makes tab characters invisible on the screen, appearing as white space.
SET TABS VISIBLE	Makes tab characters visible on the screen, appearing as a small HT (horizontal tab).

Command	Function
SET NOWRAP	Disables word wrapping at the right margin of the buffer. To start new lines, press the Return key or use the FILL command.
SET WRAP	Default setting. Enables word wrapping at the right margin of the buffer. EVE starts new lines without your pressing the Return key or using the FILL command.
UPPERCASE WORD	Changes the current word, range, or box to uppercase.

8.19 Using Buffers

8.19.1 Definition: Buffers

Buffers are storage areas that exist only during an editing session. When you edit an existing file, EVE reads the contents of the file into a buffer. The highlighted status line contains the name of the buffer, its editing status (read-only or write), editing mode (insert or overstrike), and direction (forward or reverse).

8.19.2 EVE Commands to Manipulate Buffers

The following table describes the EVE commands used to create, manipulate, and delete buffers:

Command	Function
BUFFER	Puts the specified buffer into the current window and moves the cursor to the last location it occupied in that buffer. If the specified buffer does not exist, creates a new buffer.
DELETE BUFFER	Deletes a buffer you specify by name.
GET FILE or OPEN	Puts the specified file into the current EVE window, creating a new buffer if necessary. If the file exists, EVE copies it into a new buffer in the current window. If the file does not exist, EVE creates a new, empty buffer, using the file name and file type for the buffer name. If there already is a buffer by that name, EVE asks for a different name to use.
GO TO	Returns the cursor to the location labeled by the MARK command. If the labeled location is found in another buffer, EVE moves the cursor to that buffer and puts it into the current window. (Section 8.19.10 explains how to use multiple buffers in an editing session.)
INCLUDE FILE	Inserts the contents of the specified file into the current buffer at the line above the cursor location. This is useful to combine files.
NEW	Creates a new buffer named Main and puts it into the current window. If the buffer Main already exists, EVE asks for a name for the new buffer.

Command	Function
NEXT BUFFER	Puts the next buffer (if one exists) into the current window and moves the cursor to the last position it occupied in that buffer. This command lets you move from one buffer to another without specifying a buffer name.
OPEN SELECTED	Opens a file whose name you have selected or found. This command is the same as using the GET FILE or OPEN command without having to type the file name.
REMOVE or CUT	If you are in the Buffer List buffer, same as DELETE BUFFER. Use the REMOVE command as follows to delete a buffer without typing the buffer name: enter the SHOW BUFFERS command (which puts you in the Buffer List buffer), move the cursor to the name of the buffer you want to delete, and enter the REMOVE command.
SAVE FILE	Writes the contents of the current buffer to the file associated with the buffer without ending the editing session. If you do not specify a file name with the SAVE FILE command, EVE prompts you for an output file specification. Similar to WRITE FILE.
SAVE FILE AS	Writes the contents of the current buffer to the file you specify without ending the editing session. For example, if you are editing a file named FIRST.DAT, you can save it as SECOND.TXT. This command does not change the name of the buffer. It does, however, associate the buffer with the file you name so that any subsequent SAVE FILE, WRITE FILE, or EXIT command writes the buffer to the file you named. This command requires you to supply a file specification.
SELECT or RETURN	If you are in the Buffer List buffer, selects the buffer you specify. Use the SELECT command as follows to select a buffer without typing the buffer name: enter the SHOW BUFFERS command, move the cursor to the name of the buffer you want to select, and enter the SELECT command.
SET BUFFER	Lets you specify the editing status of the buffer: whether the buffer can be modified or can be written to a file when you exit from EVE.
SHOW	Displays information about the buffers you have created during the editing session. If more than one buffer is active in your editing session, the SHOW command displays information about the buffer you are currently editing. For information about the other active buffers, press the Do key. To resume editing, press any other key.
SHOW BUFFERS	Lists the buffers you have created during an editing session. You can move the cursor through the list and specify a particular buffer for viewing by pressing the Select key.
SHOW DEFAULTS BUFFER	Shows information, such as margins, tab stops, direction, mode, and maximum lines, about the EVE system buffer named $DEFAULTS$. These are the default settings used when you create new buffers.

Command	Function
SHOW SYSTEM BUFFERS	Lists the system buffers created by EVE, such as the Message buffer, Help buffer, Insert Here buffer, and $RESTORE$ buffer. You can move the cursor through the list and specify a buffer for viewing by pressing the Select key.
WRITE FILE	Writes the contents of the current buffer to the file associated with the buffer or to the file you specify on the command line without ending the editing session. If the current buffer does not have a file specification associated with it, EVE prompts you for an output file specification. Similar to SAVE FILE.

8.19.3 Obtaining Buffer Information

To display more information about the current buffer, enter the SHOW command. The information displayed includes whether the buffer has been modified, in addition to the following:

- Buffer name
- Names of the input, output, and buffer-change journal files
- Current mode and direction
- Number of lines
- Margin and screen-width settings
- Paragraph indent
- WPS word wrap
- Wrap indent
- Tab stop

If more than one buffer is active during an editing session, EVE prompts you to press the Do key to get information about other buffers.

8.19.4 Deleting a Buffer

To delete a buffer, enter the DELETE BUFFER command and specify the name of the buffer you want to delete. If the buffer is empty or unmodified, EVE deletes it. If, however, the buffer has been modified, EVE prompts you for a choice. Note that the buffer name must be typed in full; no abbreviations are allowed. If you are viewing a buffer that you want to delete, EVE replaces the buffer with the oldest buffer existing in the editing session.

The following table lists the choices you can enter:

Keyword	Effect
DELETE_ONLY	Deletes the specified buffer.
WRITE_FIRST	Writes out (saves) the specified buffer, then deletes it.
QUIT	Default choice. The buffer is not deleted.

8.19.5 Example

In the following example, deletion of the modified buffer MYFILE.TXT is requested:

```
Command: DELETE BUFFER MYFILE.TXT
That's a modified buffer. Type delete_only, write_first, or quit:
```

8.19.6 Changing Buffer Status

Use the SET BUFFER command to change the editing status of the buffer; that is, whether the buffer can be modified and whether the buffer will be written to a file after you exit from EVE.

8.19.7 SET BUFFER Command Keywords

You can specify one of the following keywords for each command:

Keyword	Effect
MODIFIABLE	Default setting. The buffer can be modified. Also restores the previous mode of the buffer (insert or overstrike).
READ_ONLY	The buffer is *not* saved (written out) on exiting, even if it has been modified (opposite of WRITE). Also sets the buffer to unmodifiable. However, you can set it to modifiable.
UNMODIFIABLE	The buffer cannot be modified. Also overrides the mode of the buffer (insert or overstrike).
WRITE	Default setting. The buffer is saved (written out) on exiting if it has been modified (opposite of READ_ONLY). If a buffer is read-only or unmodifiable, SET BUFFER WRITE makes it modifiable and restores its previous mode (insert or overstrike).

8.19.8 Changing Buffer Status

By default, buffer status is set to MODIFIABLE and WRITE, letting you change the contents of a buffer and save the changed buffer in a file.

To change the status of a buffer so that its contents cannot be inadvertently changed, set the buffer to READ_ONLY (which implies unmodifiable) with the following command:

```
Command: SET BUFFER READ_ONLY
```

To change the status of a buffer so it becomes a temporary storage area (a ''scratchpad''), set the buffer to READ_ONLY and MODIFIABLE with the following commands:

```
Command: SET BUFFER READ_ONLY
Command: SET BUFFER MODIFIABLE
```

You then can edit the buffer, but it will not be saved when you exit from EVE.

8.19.9 Displaying the Messages Buffer

EVE uses the message window, which appears at the bottom of the screen, to communicate error and informational messages during an editing session. The message window displays the last message in the Messages buffer.

You can display these messages with the BUFFER command. To display the contents of the Messages buffer, press Do and enter the command BUFFER MESSAGES. To return to the buffer you were editing, press Do and enter the BUFFER command followed by the name of the appropriate buffer.

You can also enter the SHOW BUFFERS command to display the buffers you have created and press the Select key to choose a buffer.

8.19.10 Editing Multiple Buffers

You can use several buffers if you want to edit more than one file or if you want temporary storage areas for manipulating blocks of text. You can use one of the following commands to create a new buffer: GET FILE or OPEN, OPEN SELECTED, or BUFFER.

8.19.11 Using the GET FILE Command

To create a new buffer with a file that already exists, enter the GET FILE (or OPEN) command and the name of the file you want to copy to the new buffer. You can use the asterisk (*) wildcard character as a substitute for all or some of the characters in the file name and file type. You can use the percent sign (%) wildcard character as a substitute for one character in the file name and file type, and you can use the ellipsis ([...]) wildcard as a substitute for a directory specification.

8.19.12 Using the OPEN SELECTED Command

You can also use the OPEN SELECTED command to create a new buffer as follows:

1. Put the cursor on the name of the file you want to open.
2. Enter the OPEN SELECTED command.

8.19.13 Using the BUFFER Command

To put a specific buffer into the current EVE window, enter the BUFFER command and the name of the buffer you want to put in the current window. You cannot use wildcard characters in buffer names. The asterisk (*) and percent sign (%) are treated as literal characters in a buffer name. If the buffer you specify does not already exist, EVE creates a new buffer.

If the specified file exists, EVE reads the contents of the file into a new buffer and displays the buffer in the current window. If there is more than one match for a file specification with a wildcard, EVE displays a list of choices in the $CHOICES$ buffer and prompts you to provide a more complete file specification. EVE will open the first file it matches if you use a search list or an ellipsis ([...]) wildcard. Otherwise, EVE creates an empty buffer and displays the buffer in the current window.

To change the buffer in the current window, press the Do key, type BUFFER and the name of the buffer you want to display on the screen, and then press the Return key. If you forget a buffer name, enter the SHOW BUFFERS command to display the names of active buffers in your editing session and specify a buffer with the Select key.

8.19.14 Reading Files into EVE

There are four ways to read a file into an EVE buffer:

- Invoke EVE with a file specification.
- Enter the INCLUDE FILE command and the name of the file you want to include. EVE reads the entire contents of the file into the buffer just before the line where the cursor is located. Using the INCLUDE FILE command does not change the name of the buffer on the status line.
- Enter the GET FILE or OPEN command and the name of the file you want to use. Either command creates a new buffer and reads the contents of an existing file into the buffer. The name of the buffer on the status line is the same as the file name you specify with the GET FILE or OPEN command (see Section 8.19.10).
- Select or find a file name, then enter the OPEN SELECTED command.

8.19.15 Writing Files from EVE

To write the contents of the current buffer to a file, enter the WRITE FILE command. You can include a file specification with the WRITE FILE command. If you do not include a file specification, EVE uses the input file specification to write the file. If you created the current buffer with the BUFFER or NEW command, EVE prompts you for a file specification to which it writes the file.

8.19.16 Example

The following example shows how to use the output file associated with the buffer to write out a buffer:

```
Command: GET FILE RHYMES.DAT
     .
     .
     .
Command: WRITE FILE

3 lines written to WORKDISK:[USER]RHYMES.DAT;2
```

8.19.17 Using Windows

During an EVE editing session, the buffer you are editing is displayed on the screen in a window. A highlighted status line appears at the bottom of the window identifying the name, current editing mode, and current direction of the buffer.

EVE lets you view more than one window on your terminal screen at the same time. For example, you can have two windows on the terminal screen to view and edit different sections of the same buffer.

8.19.18 Keys Used with EVE Windows

The following table describes EVE keys used to create and manipulate windows:

Key or Key Sequence	Function in a Window Environment
GOLD Next Screen	Puts the cursor in the next (or other) window. Same as the NEXT WINDOW command. For more information about GOLD key combinations, see Section A.2.12.
GOLD Prev Screen	Puts the cursor in the previous (or other) window. Same as the PREVIOUS WINDOW command. For more information about GOLD key combinations, see Section A.2.12.

8.19.19 EVE Window Commands

The following table describes EVE commands used to create and manipulate windows:

Command	Function in a Window Environment
DELETE WINDOW	Deletes the current window, if you are using more than one window.
ENLARGE WINDOW	Enlarges the current window by a specified number of lines. For example, ENLARGE WINDOW 5 enlarges the window by five lines. The adjacent window shrinks accordingly.
NEXT WINDOW or OTHER WINDOW	Puts the cursor in the next (or other) window.
ONE WINDOW	Restores the current window as a single, large window. EVE deletes all other windows from the screen. The buffers associated with those windows are not deleted.
PREVIOUS WINDOW	Puts the cursor in the previous (or other) window.
SET WIDTH	Sets the width of lines displayed on the screen. Specify width as a positive integer. By default, the screen width is your terminal setting (usually 80 columns). If the width is set greater than 80, EVE sets the terminal to 132-column mode for the current editing session. When you exit from EVE, the terminal is restored to the default setting. Setting the width changes the display of text in all windows.
SHIFT LEFT	Moves the current window to the left a specified number of columns. You can use the SHIFT LEFT command only to reverse the effect of the SHIFT RIGHT command.
SHIFT RIGHT	Moves the current window to the right a specified number of columns, so you can view columns of characters that do not currently appear on the terminal screen.

Command	Function in a Window Environment
SHRINK WINDOW	Shrinks the current window by a specified number of lines. For example, SHRINK WINDOW 5 shrinks the window by five lines. The adjacent window expands accordingly.
SPLIT WINDOW	Splits the current window, forming two smaller windows. You can divide the window into more than two parts by specifying a number with the command. For example, SPLIT WINDOW 3 splits the window into three windows.
TWO WINDOWS	Same as the SPLIT WINDOW 2 command.

8.19.20 Viewing Two Sections of One Buffer

To view two sections of a file at the same time, use the SPLIT WINDOW command. EVE splits your screen and creates two identical windows. The cursor maintains its position in the buffer but appears only in the bottom window. The buffer name is the same in both status lines.

Displaying two sections of a long file makes moving text within a file efficient. You can select and remove text from one part of the file and insert it into the other. To move the cursor from one window to the other, enter the NEXT WINDOW command.

To remove the second window from the screen and expand the current window to occupy the whole editing area, press the Do key, enter the command ONE WINDOW, and press the Return key.

8.19.21 Editing Two Buffers

The following procedure describes how to edit two buffers containing different files:

Step	Task
1	Create two windows on your screen by entering the SPLIT WINDOW command. EVE splits your screen and creates two windows. The cursor maintains its position in the buffer but appears only in the bottom window. The buffer name in each of the highlighted status lines is the same.
2	Use the GET FILE, OPEN, or OPEN SELECTED command to put a second file in the current window. To display a buffer that you created earlier in the editing session in the current window, enter the BUFFER command and the name of the buffer you want to display. Your terminal screen now displays two different buffers. You can select and remove text from one buffer and insert it into the other buffer. To move the cursor from one window to the other, enter the command NEXT WINDOW.

8.20 Creating a Subprocess

8.20.1 Overview

You can create a subprocess to switch between an EVE editing session and DCL command level without terminating your editing session. To create a subprocess, enter the SPAWN command. EVE suspends the current editing session and connects your terminal to a new subprocess. The DCL prompt ($) appears on your terminal screen.

8.20.2 Spawning

The most common reasons to spawn a subprocess are to invoke the Mail utility and to run screen-oriented programs, although your subprocess can invoke any OpenVMS utility or execute any DCL command.

To return to your editing session, log out of the subprocess by entering the DCL command LOGOUT. EVE resumes the editing session, and the cursor appears in the location it occupied before you spawned the subprocess. You can also supply a DCL command as a parameter to the SPAWN command to create a specific subprocess.

8.20.3 Example

In the following example, the Mail utility is spawned from EVE:

```
[End of file]

Buffer: MAIN                              | Write | Insert | Forward
Command: SPAWN MAIL
```

The prompt for the Mail utility (MAIL>) appears on the screen. When you exit from Mail, you are automatically logged out of the subprocess and EVE resumes the editing session.

8.20.4 Spawning to EVE from DCL

Rather than spawn a process to use DCL, you can spawn a process for an EVE editing session and then attach to the parent DCL process to use DCL commands and utilities.

When you want to return to the DCL command level, use the EVE command ATTACH to return to the parent process.

To resume your editing session, reconnect to the editing subprocess by using the DCL command ATTACH with the process name of the subprocess.

8.20.5 Example

In the following example, a subprocess is created using the DCL command SPAWN. The SPAWN command creates the subprocess SMITH_1. At the subprocess level, EVE is invoked and the editing session is conducted. At the end of the editing session, the ATTACH command is entered and you are returned to DCL. Then, to resume the editing session,the DCL command ATTACH is entered using the the process name of the subprocess SMITH_1:

```
$ SPAWN
%DCL-S-SPAWNED, process SMITH_1 spawned
%DCL-S-ATTACHED, terminal now attached to process SMITH_1

[End of file]

Buffer: MAIN                          | Write | Insert | Forward
Command: ATTACH SMITH
$ ATTACH SMITH_1
```

9
Editing Text Files: Using EDT

9.1 Overview

Although the default text editor on the OpenVMS operating system is EVE, you may also wish to use the EDT editor. EDT is an interactive text editor with which you can create a new file, insert text into it, and modify that text. You can also edit text in existing files with EDT.

This chapter describes:

- Features of EDT
- Beginning EDT editing sessions
- Entering EDT line commands
- Entering EDT keypad commands
- Using online help in EDT
- Ending EDT editing sessions
- Changing editing modes
- Recovering from interruptions
- Summary of EDT commands

9.1.1 References

For more information on using the EDT editor, refer to *OpenVMS EDT Reference Manual*.

9.2 EDT Overview

9.2.1 Editing Modes

EDT provides both line editing and **keypad** editing. In line editing, you type the editing command and the range of text you want the command to affect. In keypad editing, you move the cursor directly to the text you want to change and press keypad keys to enter the editing commands.

One way to use EDT is to use the keypad as the primary editing mode in combination with various line-editing commands.

9.2.2 Nokeypad Editing

You can also redefine certain keypad and control keys to perform editing functions not available in keypad mode. This is called nokeypad editing. Nokeypad commands are the basis for keypad mode key definitions and consist of English words and abbreviations. You can use nokeypad mode commands to define keys.

9.3 Beginning EDT Editing Sessions

9.3.1 Invoking EDT

To invoke EDT, enter the DCL command EDIT/EDT followed by the name of the file you want to edit. The first few lines of the latest version of the file appear on the screen. The cursor is positioned at the top of the screen, and EDT is ready to receive a keypad-editing command.

9.3.2 Example

In the following example, a file named MEMO.TXT is edited:

```
$ EDIT/EDT MEMO.TXT

Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
```

9.3.3 Editing Existing Files

When you edit a file that already exists (for example, if you created the file during a previous session), EDT saves the existing versions and places a copy of the latest version in your buffer. A buffer is the temporary storage area in which you edit text. The existing versions of the file remain unchanged.

9.3.4 Creating Files

If you invoke EDT to create a file, the following message appears:

```
$ EDIT/EDT NEWFILE.TXT

Input file does not exist
[EOB]
*
```

Only the EDT message and the end-of-buffer symbol ([EOB]) appear on the screen, and EDT is ready to receive keypad-editing commands.

9.3.5 Changing Editing Modes

In the previous examples, you enter EDT in keypad (change) mode because a startup command file (SYS$LOGIN:EDTINI.EDT) containing the SET MODE CHANGE command has been executed. If this command is not executed in an EDT startup command file, you enter EDT in line mode. Enter the CHANGE command at the asterisk (*) prompt to change to keypad mode.

For information on creating a startup command file, see the *OpenVMS EDT Reference Manual*.

9.4 Entering EDT Line Commands

9.4.1 Line Editing

EDT prompts for line-editing commands with an asterisk (*). Line-editing commands usually operate on a range of one or more lines of text that you specify as a parameter for the command. You can abbreviate EDT line-editing commands. For clarity, the examples in this chapter show complete line-editing commands.

9.4.2 Example: Line Editing

The following example shows the line editing command that you would enter to display an entire file on your screen:

```
*TYPE WHOLE
```

9.4.3 Using Line Numbers

To help you locate and edit text, EDT assigns line numbers. These line numbers are not part of the text and are not kept when you end an editing session.

When you insert new text, EDT numbers the lines using decimal numbers. For example, if you add a line of text between lines 13 and 14, it is numbered 13.1. To avoid confusion when working with decimal numbers, enter the RESEQUENCE command. The RESEQUENCE command renumbers all the lines from the cursor to the end of the buffer in increments of one.

Note that the EDT line-editing command SET NUMBERS (the default) must be in effect for line numbers to be displayed in EDT line editing.

9.4.4 Example: Displaying Line Numbers

The following example shows how to display all the line numbers in an existing file:

```
* TYPE WHOLE
    1          oneoneoneoneoneoneone
    2          twotwotwotwotwo
    3          threethreethree
    4          fourfourfourfour
    5          fivefivefivefive
[EOB]
*
```

9.4.5 Specifying a Range of Lines

Some EDT line-mode commands can affect a range of lines. For example, the INSERT command will create a new line of text in your buffer; to insert a new line of text at the beginning of your buffer, enter the command INSERT BEGIN.

9.4.6 EDT Line-Mode Command Ranges

The following table describes the different ranges you can specify when you edit a file in line mode:

Range Type	Description
period (.)	Current line
number	EDT line number
'string'	Next line containing the quoted string
BEGIN	First line of the buffer
END	After the last line in the buffer ([EOB])
LAST	Last line EDT was at in the previous buffer
WHOLE	Entire buffer
BEFORE	All lines in the buffer before the current line
REST	All lines in the buffer starting with the current line and ending with the last line

9.4.7 EDT Command-Line Symbols for Specifying Ranges

The following table lists symbols and words that you can combine with the line-mode command ranges:

Symbol or Word	Description
, or AND	Used to join noncontiguous ranges in a list; only single lines can be joined in this way
: or THRU	Indicates a group of lines starting with the first range specifier and ending with the second
n	Indicates the number of lines from the current line
# n or FOR n	Indicates the next *n* number of lines
+ ''string'' or ''n''	Indicates that *string* or *n* refers to a line or lines after the current line
– ''string'' or ''n''	Indicates that *string* or *n* refers to a line or lines before the current line
ALL ''string'' or ''n''	Indicates that the command applies to all lines containing *string*

9.4.8 Canceling EDT Commands

Use Ctrl/C to cancel the currently executing EDT command without affecting previous edits. For example, to stop the display of a long file, press Ctrl/C.

```
*TYPE WHOLE
 .
     .
     .
Ctrl/C
Cancel
Press return to continue
Return
Aborted by CTRL/C
```

9.5 Entering EDT Keypad Commands

9.5.1 Overview

While line editing allows you to manipulate large portions of text easily, keypad editing provides easy manipulation of small units of text. EDT keypad commands enable you to find, insert, delete, substitute, and move text in a file. The cursor can be moved through a file in a variety of ways. The position of the cursor in a file determines how text will be affected by EDT commands.

9.5.2 Keypad Editing

In keypad editing, the screen displays editing changes as you make them. You type text from the main keyboard and enter keypad-editing commands from the numeric keypad. To display a diagram of the keypad keys, press PF2 while in keypad mode. To initiate keypad editing, you must first enter the line-editing command CHANGE or have SET MODE CHANGE in your EDT startup file. See Section 9.8.3 for information on the CHANGE command.

In keypad mode, you can manipulate the cursor with commands that move it unit-by-unit through the text or with commands that move it directly to a particular location.

9.5.3 Using the Keypad

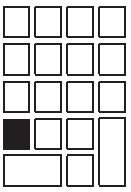
Each key in the keypad performs at least one editing command; most perform two. Pressing a key invokes the primary function. To invoke the alternate function of a key, press the GOLD key (labeled PF1) first, then press the desired key. In the examples that follow, a small diagram of the keypad highlights the key or key sequences that perform the command being described. The text associated with the keypad illustrates the effect of that editing command.

The supplemental editing keys on the keypad perform the same functions as some of the EDT keypad keys.

9.5.4 Example: Using the WORD Function

Keypad key 1 (KP1) performs both the WORD and the CHNGCASE functions. To enter the WORD command, press KP1. The cursor moves to the beginning of the next word.

WORD



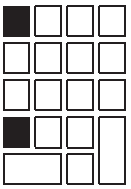
```
Once the weather turns cold, mice may
find a crack in your foundation and enter your house.  They're
looking for food and shelter from the harsh weather ahead.

[EOB]
```

9.5.5 Example: Using the CHNGCASE Function

To enter the CHNGCASE command, press the GOLD key first and then CHNGCASE. The character at the cursor (or the characters highlighted with the Select key) changes from lowercase to uppercase or from uppercase to lowercase.

CHNGCASE



```
Once The weather turns cold, mice may
find a crack in your foundation and enter your house.  They're
looking for food and shelter from the harsh weather ahead.

[EOB]
```

The ‘‘T’’ in the word ‘‘The’’ is now capitalized.

9.6 Using Online Help in EDT

9.6.1 Getting Keypad Help

In keypad mode, you can display a diagram of the keypad keys by pressing PF2. On LK201-series keyboards, you can also use the Help key on the supplemental editing keypad. To display information about a particular keypad command, first press the Help key and then press the keypad key.

9.6.2 Getting Line Mode Help

To request help in EDT while in line mode, enter the HELP command at the asterisk (*) prompt and press Return. To display information about a particular command, type HELP followed by the name of the command. EDT displays information about the command and lists related topics. For example, to request help on the COPY command, enter the following command line:

```
*HELP COPY Return
```

9.6.3 Getting Nokeypad Help

If you are in nokeypad mode and want to get help information about nokeypad commands, enter HELP CHANGE at the asterisk (*) prompt.

9.7 Ending EDT Editing Sessions

9.7.1 Overview

To terminate an EDT session, press Ctrl/Z. This puts you into line-editing mode. You can type EXIT or QUIT at the asterisk (*) prompt. EXIT saves your edits in a new version of the file; QUIT terminates the editing session and does not save your edits.

The existing versions of a file remain unchanged regardless of how you end the editing session. To override the default output file name, enter the EXIT command with a new file specification as the parameter. Note that if a file is given the same file name as an existing file, the two files will have the same file name and file type, but different version numbers and content.

9.7.2 Saving Edits

By default, the EXIT command creates an output file with the same file name and file type as the input file but with the version number incremented by 1.

9.7.3 Examples

- In the following example, the EXIT command is entered after a file named MEMO.TXT;3, is edited. EDT then creates a higher version named MEMO.TXT;4:

```
*EXIT
DISK1:[USER]MEMO.TXT;4  2 lines
$
```

- If the same editing session is ended with the command EXIT MICE.TXT, EDT names the output file MICE.TXT;1, provided no other file named MICE.TXT exists. If a file named MICE.TXT exists, EDT names the output file MICE.TXT; version-number, where *version-number* is one greater than the highest version number of an existing MICE.TXT file.

9.7.4 Ending EDT Sessions Without Saving Edits

To terminate EDT without saving your edits, use the line-editing command QUIT. All edits you have made to the text are ignored, and no output file is created.

The QUIT command is a useful way to terminate EDT when you have opened a file by mistake. No new file version is created.

9.8 Changing Editing Modes

9.8.1 Overview

You can switch back and forth between line and keypad editing. You can also enter line-editing commands from keypad mode. If find yourself frequently returning to line mode to enter EDT commands, you might find it easier to work in line mode. For example, if you are examining a file on a line-by-line basis, using

line numbers as reference points, line-mode editing is more appropriate. In contrast, if you need to examine, cut, and paste large chunks of text between nonadjacent areas within a file or between two files, keypad editing might be faster.

9.8.2 Changing from Keypad to Line Editing

To change from keypad editing to line editing, press Ctrl/Z. When you see the asterisk (*) prompt at the bottom of your screen, enter a line-editing command at the prompt. For example:

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
Ctrl/Z


* INSERT
```

9.8.3 Changing from Line to Keypad Editing

To change from line editing to keypad editing, enter the CHANGE command:

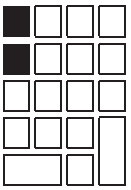
```
*CHANGE
```

The first 22 lines of the file display on your screen. If the file has fewer than 22 lines, the [EOB] symbol appears below the last line of the file.

9.8.4 Entering Line-Editing Commands from Keypad Mode

The keypad COMMAND function allows you to enter line-editing commands without leaving keypad mode. First, enter the COMMAND function by pressing the GOLD key (PF1) and then the COMMAND key (KP7).

COMMAND



EDT displays the Command: prompt. At the prompt, enter a line-editing command.

9.8.5 Example

The following example enters the line-editing command SET QUIET, which suppresses the sound made when EDT issues an error message. To execute the command, press the Enter key. (If you press Return by mistake, ^M appears; delete the ^M by pressing the Delete key on the main keyboard and press Enter.)

```
Once the weather turns cold, mice may find a crack in your
foundation and enter your house.  They're looking for food and
shelter from the harsh weather ahead.

[EOB]
Command:   SET QUIET
```

9.9 Recovering from Interruptions

9.9.1 Restoring the Display

Pressing Ctrl/W removes extraneous characters (such as a broadcast message or a message indicating that you have received electronic mail) from the screen and restores the previous display. Use Ctrl/W to ensure that the cursor is in the correct position.

9.9.2 Recovering from Ctrl/Y

The DCL command CONTINUE resumes an editing session that was interrupted by pressing Ctrl/Y, as long as only built-in DCL commands were entered after pressing Ctrl/Y. For example, you could press Ctrl/Y, enter the command SHOW TIME, and return to your editing session with the CONTINUE command. You enter the SHOW TIME and CONTINUE commands at the DCL prompt.

After you enter the CONTINUE command, press Ctrl/W to refresh the screen display. EDT redisplays the text of your editing session.

9.9.3 Journal Files

By default, EDT keeps a journal file with the same file name as the input file and a file type .JOU. If the editing session ends without interruption, the journal file is deleted when you terminate the session. If the editing session is aborted (for example, during a system failure, in response to pressing Ctrl/Y, or entering the QUIT/SAVE command), you can recover your edits with the exception of those commands entered just prior to the interruption. Enter the same command line you used to begin the editing session, adding the /RECOVER qualifier. For example:

```
$ EDIT/RECOVER MICE.TXT
```

EDT will reproduce the editing session, reading the commands from the journal file and executing them on the screen.

9.10 Summary of EDT Commands

9.10.1 Changing Editing Modes

The following table describes commands and keys that can be used to change edit modes:

Keypad Mode	Line Mode	Nokeypad Mode	Description
COMMAND		EXT (Extend)	Enables you to enter a line-mode command while EDT is still in keypad or nokeypad mode.
Ctrl/Z	CHANGE	EX	Transfers your editing session from one mode (line, keypad, or nokeypad) to another.
	SET MODE		Establishes the initial mode of the EDT session when used in a startup command file.

Keypad Mode	Line Mode	Nokeypad Mode	Description
	SHOW MODE		Indicates which SET MODE command was most recently issued.

9.10.2 Moving the Cursor

The following table describes commands and keys that can be used to move the cursor:

Keypad Mode	Line Mode	Nokeypad Mode	Description
BACKSPACE		BL	Moves the cursor to the beginning of the current line.
BOTTOM	TYPE END	ER	Moves the cursor to the end of the buffer, after the last character position in the buffer.
CHAR		C	Moves the cursor one character in the current direction (forward or backward, depending on whether ADVANCE or BACKUP is in effect).
↓		↓	Moves the cursor down one line toward the bottom of the buffer, regardless of whether ADVANCE or BACKUP is in effect.
EOL		EL	Moves the cursor to the end of the current line if the direction is forward. If the current direction is backward, the cursor moves to the end of the previous line.
←		←	Moves the cursor one character to the left, regardless of whether ADVANCE or BACKUP is in effect.
		‘‘move’’	Moves the cursor within the current buffer.
LINE		L	Moves the cursor to the beginning of the next line if the direction is forward or to the beginning of the current line if the direction is backward. If the direction is backward, pressing LINE again moves the cursor to the beginning of the previous line.
		KS	Modifies the position of the cursor at the completion of the PASTE command.
PAGE		PAGE PAGETOP	Moves the cursor to the right of the next page marker or to the next form-feed character. If you have no page markers (defined with the SET ENTITY PAGE command), the PAGE entity is the whole buffer.

Keypad Mode	Line Mode	Nokeypad Mode	Description
		TOP	Moves the cursor to the top of the screen.
→		→	Moves the cursor one character to the right, regardless of of whether ADVANCE or BACKUP is in effect.
SECT		16L.	Moves the cursor one section (16 lines) toward the end or the beginning of the buffer, depending on whether ADVANCE or BACKUP is in effect.
	SET CURSOR		Controls scrolling of the screen relative to the cursor position. This command has no effect if you are editing in line mode.
	SHOW CURSOR		Displays values set by the SET CURSOR command.
TOP		BR	Moves the cursor to the first character at the beginning of the buffer.
↑		↑	Moves the cursor up one line toward the top of the buffer regardless of of whether ADVANCE or BACKUP is in effect.
WORD		W	Moves the cursor to the beginning of the next word in the current direction (forward or backward, depending on whether ADVANCE or BACKUP is in effect).

9.10.3 Inserting Text

The following table describes commands and keys that can be used to insert text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
Ctrl/L		^L.	Inserts a form-feed character (<FF>) into your text.
Ctrl/M		^M.	Inserts a carriage-return character (<CR>) into your text.
Ctrl/R	Ctrl/R	REF	Clears and redraws the screen display (in keypad mode) or line (in line mode), eliminating any extraneous characters or messages. The current text you are editing remains unchanged. In keypad mode, Ctrl/R is identical to Ctrl/W.
Ctrl/W		REF	Clears and redraws the screen display (in keypad mode) or line (in line mode), eliminating any extraneous characters or messages. The current text you are editing remains unchanged.

Keypad Mode	Line Mode	Nokeypad Mode	Description
	Ctrl/Z	Ctrl/Z	Completes the insert operation and returns EDT to the command state. Used with the INSERT (nokeypad I) and REPLACE (nokeypad R) commands.
FILL (VT100) Ctrl/F (VT52)	FILL	FILL FILLSR	Takes a select range of lines and reorganizes the text so that the maximum number of whole words can fit within the current line width. In line mode, fills a selected range of lines.
	INCLUDE		Copies external files into the EDT text buffer. In line mode, EDT displays an asterisk (*) prompt when the INCLUDE command finishes copying the file. In keypad or nokeypad mode, the included text appears on the screen.
OPEN LINE	INSERT	I Return	Inserts a line terminator in the text you are editing at the current cursor position and makes the line terminator the new cursor character.
	INSERT	I	Adds text to the current or specified buffer.
SPECINS		ASC (ASCII) Circumflex (^)	Enables you to insert any character from the DEC Multinational character set into your text, using the character's decimal equivalent value (see Appendix B). The circumflex (^) works only for characters with decimal values 0 to 31.

9.10.4 Deleting and Restoring Text

The following table describes commands and keys that can be used to delete and restore text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
Ctrl/U		DBL	Deletes everything from the character to the left of the cursor to the beginning of the line.
	DELETE	D	Deletes a line or group of lines, depending on the range that you specify. If you do not specify a buffer or a range, EDT deletes the current line. If you specify a buffer but not a range, EDT moves to that buffer and deletes its entire contents.
DEL C	DELETE	DC	Deletes the character on which the cursor is positioned.
		D-C	Deletes the character to the left of the cursor.

Keypad Mode	Line Mode	Nokeypad Mode	Description
		D+C	Deletes the character to the right of the cursor.
DEL EOL	DELETE	DEL	Deletes everything on a line from the current cursor position up to, but not including, the line terminator.
DEL L	DELETE	D+NL	Deletes everything on a line from the current cursor position up to and including the line terminator.
DEL W	DELETE	DEW	Deletes words or parts of words.
LINEFEED		DBW	Deletes the word or characters in a word to the left of the cursor up to the beginning of the previous word.
REPLACE	REPLACE	R (Replace)	In keypad mode, deletes text in the select range and replaces it with the contents of the PASTE buffer. In line and nokeypad mode, deletes the lines specified by range from the current or specified buffer and replaces the deleted lines with text that you enter at the terminal.
UND C		UNDC	Inserts the current contents of the delete character buffer into text to the left of the cursor.
UND L		UNDL	Inserts the current contents of the delete line buffer into text to the left of the cursor.
UND W		UNDW	Inserts the current contents of the delete word buffer into text to the left of the cursor.

9.10.5 Locating Text

The following table describes commands and keys that can be used to locate text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
ADVANCE		ADV	Sets the direction for subsequent editing work to forward (to the right of the cursor and down toward the end of the buffer).
BACKUP		BACK	Sets the direction for subsequent editing work to backward (to the left of the cursor and toward the beginning of the buffer).
		CLSS	Clears the text string currently in the search buffer.
FIND	FIND	‘‘string’’ ^@.	Searches for specified text.

Keypad Mode	Line Mode	Nokeypad Mode	Description
FNDNXT	FIND ""	""	Searches for the next occurrence of a string defined by the FIND command.
RESET		RESET	Cancels the active select range, sets the direction to advance, and sets EDT to the DMOV (default move) state.
		DESEL	Cancels the active select range.
		TGSEL	Combines the SEL and DESEL commands. When there is an active select range, the TGSEL command cancels it, performing the same function as the DESEL command. When there is no active select range, TGSEL initiates the process of creating a select range, just as the SEL command does.
SELECT		SEL	Sets up a select range for use with keypad functions such as APPEND, CHNGCASE, CUT, FILL, REPLACE, SUBS, and Ctrl/T.
		SSEL	Finds a string and designates it as a select range.
	SET SEARCH		Determines how EDT locates strings during your editing sessions.
	SHOW SEARCH		Indicates the search parameters EDT uses to locate strings in text.

9.10.6 Substituting Text

The following table describes commands and keys that can be used to substitute text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
SUBS	SUBSTITUTE	S	In keypad mode, replaces the current search string with the contents of the PASTE buffer. In line and nokeypad mode, replaces one string with another throughout the specified range.
	SUBSTITUTE NEXT	SN	Searches for the next occurrence of a string and replaces it with another string. This command uses strings that have been stored in the search buffer and in the substitute buffer.

9.10.7 Moving Text

The following table describes commands and keys that can be used to move text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
APPEND		APPEND	Deletes the select range (keypad mode) or specified entity (nokeypad mode) from the current buffer and adds it to the end of either the PASTE buffer (the default) or the specified buffer.
	COPY		Copies the specified text to the specified location. You can copy a range of text from one location to another within the same buffer, or you can copy to and from different buffers, creating new buffers as appropriate. No text is deleted. The /DUPLICATE qualifier enables you to copy the specified text *n* times.
CUT		CUT	In keypad mode, removes the active select range from the current buffer and stores it in the PASTE buffer. In nokeypad mode, removes the specified entity from the text buffer and stores it in another specified buffer.
CUT + PASTE	MOVE	CUT + PASTE	Moves lines from one location to another within the current buffer or from one buffer to another. The lines are deleted from their original position and are inserted at the new location.
PASTE	COPY	PASTE	Copies or moves text within a buffer. In keypad mode, PASTE copies the PASTE buffer contents into the current buffer. In nokeypad mode, PASTE copies the contents of any buffer into the current buffer.
	MOVE		

9.10.8 Indenting Text

The following table describes commands and keys that can be used to indent text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
Ctrl/A		TC (Tab Compute)	Establishes a tab position and resets the indentation level. The indentation level is the number of columns, starting at the left of the screen, that you want to leave blank before beginning a line of text. To use this command, the current cursor position must be a multiple of the SET TAB value.
Ctrl/D		TD (Tab Decrement)	Decreases the current indentation level count by one setting. The indentation level count is generally set by the Ctrl/A or TC (Tab Compute) command.
Ctrl/E		TI (Tab Increment)	Increases the current indentation level count by one setting. The indentation level count is generally set by the Ctrl/A or TC (Tab Compute) command.

Keypad Mode	Line Mode	Nokeypad Mode	Description
Ctrl/T	TAB ADJUST	TADJ (Tab Adjust)	Uses the value established by the line-mode SET TAB command to indent lines of text in a select range. Requires SET TAB to be in effect.
	SET TAB SHOW TAB		Establishes the SET TAB value for the various tabbing functions (tab compute, tab adjust, tab increment, and tab decrement). SHOW TAB indicates the SET TAB value and the tab indentation level count.
		SHL (Shift Left)	Moves the entire buffer text eight characters (one tab stop) to the left.
		SHR (Shift Right)	Moves the entire buffer text eight characters (one tab stop) to the right.
TAB Ctrl/I		TAB	Moves text to the right, regardless of whether ADVANCE or BACKUP are in effect. The number of column positions that the text moves depends on the cursor position, the value set by the SET TAB command (if one is in effect), and the indentation level count (if one is in effect).

9.10.9 Changing the Case of Text

The following table describes commands and keys that can be used to change the case of text:

Keypad Mode	Line Mode	Nokeypad Mode	Description
CHNGCASE		CHGC	Changes the case of letters in your text. Uppercase letters become lowercase; lowercase letters become uppercase.
		CHGL	Changes all uppercase letters within the specified entity to be lowercase. Letters that are already lowercase remain unchanged.
		CHGU	Changes all lowercase letters within the specified entity to be uppercase. Letters that are already uppercase remain unchanged.
		DLWC	Changes uppercase letters to lowercase wherever the cursor is moved.
		DUPC	Changes lowercase letters to uppercase wherever the cursor is moved.

Keypad Mode	Line Mode	Nokeypad Mode	Description
		DMOV	Returns the editing session to the default state after you use either DLWC (default lowercase) or DUPC (default uppercase).
	SET CASE SHOW CASE		Uses flags to distinguish uppercase and lowercase letters at a single-case terminal. SHOW CASE indicates which case (upper, lower, or none) has been established by the SET CASE command.

9.10.10 Using Multiple Buffers

The following table describes commands and keys that can be used when using multiple buffers:

Keypad Mode	Line Mode	Nokeypad Mode	Description
	CLEAR		Deletes the contents of the specified buffer.
	SHOW BUFFER		Lists all accessible buffers currently in your EDT session.

9.10.11 Defining Keys

The following table describes commands and keys that can be used to define keys:

Keypad Mode	Line Mode	Nokeypad Mode	Description
		BELL	Causes the terminal bell to sound when a command is processed. Used primarily in keypad key definitions.
Ctrl/K	DEFINE KEY	DEFK	Defines or redefines function keys used in keypad editing. Key definitions are based on nokeypad commands. In keypad mode, Ctrl/K starts the key definition process. In nokeypad mode, you can define a key sequence other than Ctrl/K to handle the key definition process.
	SHOW KEY		Displays the definition of any keys that have keypad editing functions.
	SET [NO]KEYPAD SHOW KEYPAD		Determines which screen editing mode (keypad or nokeypad) EDT accesses from line mode when you enter the CHANGE command. SHOW KEYPAD indicates which mode is in effect.

9.10.12 Controlling Screen and Terminal Settings

The following table describes commands and keys that can be used to control the EDT screen and terminal settings:

Keypad Mode	Line Mode	Nokeypad Mode	Description
Return			Adds a line terminator to the left of the current cursor position in the text you are editing.
	SET [NO]AUTOREPEAT		Prevents keypad keys (including arrow keys) from repeating faster than EDT can update the screen.
	SHOW AUTOREPEAT		Indicates whether autorepeat is in effect.
	SET LINES		Limits the number of lines that EDT displays on the terminal screen at one time.
	SHOW LINES		Displays the line limit.
	SET [NO]NUMBERS		Determines whether EDT displays line numbers during line-mode editing.
	SHOW NUMBERS		Displays the current setting.
	SET PARAGRAPH [NO]WPS		Sets paragraph default boundary limits.
	SHOW PARAGRAPH		Indicates whether SET PARAGRAPH WPS or SET PARAGRAPH NOWPS is in effect.
	SET [NO]QUIET		Silences the terminal bell that ordinarily sounds whenever EDT displays an error message during a screen-mode editing session.
	SHOW QUIET		Indicates whether the bell has been turned off.
	SET [NO]REPEAT		Disallows use of the GOLD key repeat feature, which allows you to repeat functions in keypad mode, and the SPECINS keypad function.
	SHOW SCREEN		Displays the current screen width setting.
	SET TERMINAL SHOW TERMINAL		Corrects or changes terminal settings to match the type of terminal you are using. SHOW TERMINAL displays the terminal settings that are currently in effect for your editing session.
	SET TERMINAL		Corrects or changes terminal settings to match the type of terminal you are using.
	SHOW TEXT		Indicates what text is displayed for the form-feed character or the end-of-buffer mark.
	SET [NO]TRUNCATE		Causes lines longer than the current screen width to wrap onto subsequent lines when you are working in screen mode. (In line mode, EDT always wraps long lines.) SET TRUNCATE does not take word boundaries into consideration; enter SET WRAP to break lines at word boundaries.
	SHOW TRUNCATE		Indicates whether SET TRUNCATE is in effect.

Keypad Mode	Line Mode	Nokeypad Mode	Description
	SET WORD [NO]DELIMITER		Determines how word entity boundaries are interpreted by EDT. By default, word **delimiters** are treated as separate words (SET WORD DELIMITER).
	SHOW WORD		Indicates whether SET WORD NODELIMITER is in effect.
	SET [NO]WRAP		Causes lines of text to wrap when new text is inserted into a buffer in keypad mode. The SET WRAP command also determines the line length for the FILL command.
	SHOW WRAP		Indicates whether the SET WRAP command is in effect and, if so, what the SET WRAP value is.

9.10.13 Processing EDT Commands

The following table describes commands and keys that can be used to process EDT commands:

Keypad Mode	Line Mode	Nokeypad Mode	Description
Ctrl/C	Ctrl/C	Ctrl/C	Interrupts certain operations (such as a search through a long file) before EDT finishes processing them.
Do (LK201 only)	Return	Period (.)	Processes searches and line editing commands.
Enter	Return	Return	Processes searches, line editing commands, and key definitions.
	SET ENTITY		Defines the delimiters that mark the word, sentence, paragraph, and page boundaries for commands and functions.
	SHOW ENTITY		Lists the current delimiters.
	SET COMMAND		Processes additional startup command files at the beginning of your EDT session. This command is valid only in an EDT startup command file.
	SHOW COMMAND		Displays the name of the active startup command file. This command is valid only in an EDT startup command file.
	SET [NO]VERIFY SHOW VERIFY		Displays the commands in a startup command file or EDT macro as the commands are processed. SHOW VERIFY indicates whether SET VERIFY is in effect.

9.10.14 Other EDT Commands

The following table describes miscellaneous commands and keys available in EDT:

Keypad Mode	Line Mode	Nokeypad Mode	Description
		DATE	Inserts the current date into your text.
	DEFINE MACRO		Creates new line-mode commands for the duration of your editing session.
	EXIT		Creates an external file, copies the contents of the MAIN buffer into that file, and ends the editing session.
GOLD			Performs various editing functions when used with other keypad and keyboard keys.
Help	Help	Help	In keypad and line modes, accesses the EDT Help utility. In nokeypad mode, defines a different key or key sequence in keypad mode to carry out the keypad Help function.
	PRINT		Copies the specified range of lines or specified buffer to an external file in a specified directory. EDT adds a form feed and two blank lines for every 60 lines it copies. The EDT line numbers become part of the text in the external file.
	QUIT	QUIT	Ends the session without copying text to an external file.
	RESEQUENCE		Assigns new EDT line numbers to the lines of the current or specified buffer.
	SHOW FILES		Displays the current input file and output file for your EDT session.
	SET [NO]FNF		Suppresses the message that appears when you use EDT to create a new file (FNF stands for File Not Found). This command is used only in startup command files.
	SHOW FNF		Indicates whether SET FNF or SET NOFNF is in effect. This command is used only in startup command files.
	SET HELP		Enables you to access different help files for your EDT session.
	SHOW HELP		Displays the name of the help file currently available for your editing session.

Keypad Mode	Line Mode	Nokeypad Mode	Description
	SET [NO]SUMMARY SHOW SUMMARY		Suppresses summary information displayed when you enter the EXIT or WRITE commands. By default, EDT displays the complete file specification and number of lines in the file that EDT has created as a result of entering the EXIT or WRITE command. SHOW SUMMARY indicates whether the SET SUMMARY command is in effect.
	SHOW VERSION		Displays the version of EDT that is being used by your operating system.
	TYPE		Displays lines of text at your terminal.
	WRITE		Copies text from an EDT buffer to an external file.
		XLATE	Passes information back to the calling program. You can enter this command when EDT has been called by a running program.

10

DIGITAL Standard Runoff (DSR): Formatting Text Files

10.1 Overview

DIGITAL Standard Runoff (DSR) is a text-formatting facility. This chapter describes:

- Entering DSR commands
- Invoking DSR
- Creating tables of contents
- Creating indexes
- Summary of DSR commands

10.1.1 References

For additional information on DSR, refer to the *OpenVMS DIGITAL Standard Runoff Reference Manual* or DCL Help for a complete description of the RUNOFF commands and qualifiers.

10.2 About DSR

10.2.1 Overview

By inserting DSR commands, control characters, and other special identifiers within a text file, you can use DSR to:

- Determine the size of pages
- Create uneven or justified right margins
- Place space between lines
- Form lists
- Create title pages, footnotes, tables of contents, indexes, and appendixes

10.2.2 Formatting a File Using DSR

The steps for formatting a file with DSR are as follows:

Step	Task
1	Create the source file with EDT, EVE, or another text editor. By default, the DSR source file has the file type .RNO.
2	Enter DSR commands, flags, and control characters within the source file to indicate how the file is to be formatted. DSR flags are special characters that you insert in text to specify emphasis of text, case of characters, spacing of characters, and so forth.
3	Process the file with the DCL command RUNOFF. When DSR processes the source file, the DSR commands cause the text to be formatted into sections, paragraphs, lists, and so on. Neither the DSR commands nor the DSR flags appear in the final document.

10.3 Entering DSR Commands

10.3.1 Creating Source Files

To enter a DSR command, create the source file with EDT, EVE, or another text editor. Begin the command in column 1 of a line and precede the command with a period. Most DSR commands have standard abbreviations. For example, you can abbreviate the .NO CONTROL CHARACTERS command as .NCC.

10.3.2 Example

The following example shows how to insert a blank line between two lines of text:

```
We sail the ocean blue,
.BLANK
And our saucy ship's a beauty.
```

10.4 Invoking DSR

10.4.1 Using the RUNOFF Command

After you add DSR commands to your file and exit from the editor, you are ready to process the file with DSR. To invoke DSR, enter the RUNOFF command followed by the name of the file you want to process.

If you process a file with the file type .RNO, you need only to enter the file name, not the file type. By default, the RUNOFF command produces an output file with the same file name as the input file and the file type .MEM.

10.4.2 Examples

Both of the following examples produce output files named FUN.MEM.

- In the following example, a file named FUN.RNO is processed:

```
$ RUNOFF FUN
```

- In the following example, a file named FUN.FUN is processed (both the file name and the file type are specified):

```
$ RUNOFF FUN.FUN
```

10.4.3 Overriding DSR Commands or Flags

By using qualifiers with the RUNOFF command, you can override DSR commands or flags included in your text file. RUNOFF command qualifiers allow you to alter the position of the text on all pages of the document, to specify emphasis such as underlining and bolding, and to otherwise control the appearance of printed output.

10.4.4 Example

In the following example, the /NOBOLD qualifier suppresses any bolding specified in the file by using the DSR command .FLAGS BOLD:

```
$ RUNOFF/NOBOLD FUN
```

10.4.5 RUNOFF Command Qualifiers

The following table summarizes the RUNOFF command qualifiers:

Qualifier	Description
/BACKSPACE	Uses the Backspace character to bold, overstrike, or underline text as it is printed. This generally gives more exact underlining and bolding for files printed on letter-quality printers. The /BACKSPACE qualifier is not recommended for **line printers**.
/[NO]BOLD	Enables and disables bolding. Any bolding specified in chapter and header titles appears in the table of contents.
/[NO]CHANGE_BARS	Enables and disables the appearance of change bars in the output file.
/CONTENTS	Generates a table of contents. (See Section 10.5.)
/[NO]DEBUG	Traces the operation of certain DSR commands by causing the commands to appear in the output file.
/DEVICE	Specifies printing options.
/DOWN	Specifies the number of blank lines to be inserted at the top of each page, preceding any header information.
/INDEX	Generates an index. (See Section 10.6.)
/FORM_SIZE	Controls the number of lines that can be accommodated per page of output.

Qualifier	Description
/[NO]INTERMEDIATE	Generates an intermediate binary file with the default file type .BRN for use with the DSR Table of Contents utility and the DSR Indexing utility.
/[NO]LOG	Controls whether or not DSR displays processing information at your terminal.
/MESSAGES	Lets you specify whether you want error messages displayed on your terminal or in an output file only. By default, DSR displays messages in both places.
/[NO]OUTPUT	Specifies the name of the output file produced by DSR.
/PAGES	Limits the output file to a specified range of pages.
/[NO]PAUSE	Controls whether DSR pauses after printing each page of output.
/REVERSE_EMPHASIS	Specifies that underlining of flagged text is to be done after the text is printed. By default, the printer prints the underscores, issues a carriage return without a line feed, then prints the flagged text above the underscores.
/[NO]RIGHT	Causes the text on each page to be shifted to the right.
/SEPARATE_UNDERLINE	Underlines text by using separate characters on the next line instead of overprinting with underscores on the same line.
/[NO]SEQUENCE	Controls whether DSR outputs line numbers from the input file.
/[NO]SIMULATE	Controls whether blank lines or form feeds are used to advance to the top of each page.
/[NO]UNDERLINE_CHAR	Allows you to specify the character to be used for underlining of flagged text.
/VARIANT	Controls the execution of the condition commands (.IF, .IFNOT, .ELSE, .ENDIF) by specifying the names of the segments to be processed.

10.4.6 Using DSR Defaults

When you use DSR to process a file, your output file looks different from your input file because DSR provides the following standard format default settings:

- A standard typewriter page size of 8 1/2 x 11 inches; that is, a width of 70 character positions and a length of 58 lines of text per page (.PAGE SIZE 58,70)
- Sequential page numbering for every page but the first (.PAGING)
- A left margin setting of 0 (just before the first character position of a line) and a right margin setting of 70 (just after the 70th character position of a line) (.LEFT MARGIN 0 and .RIGHT MARGIN 70)

- Line spacing equivalent to the single-space setting on a typewriter (.SPACING 1)
- A tab setting every eighth character position on a line (.TAB STOPS 8, 16, 24, ...)
- Filling: DSR fills each line with as many words as possible until the addition of another complete word would exceed the right margin (.FILL)
- Justification: DSR adds spaces between words to expand each line exactly to the right margin, making the right margin even or justified (.JUSTIFY)

10.4.7 Disabling Default Settings

If you do not want your file to be formatted according to the DSR default commands (shown in parentheses in the preceding list), you must disable them. See the *OpenVMS DIGITAL Standard Runoff Reference Manual* for a complete list of the default commands provided by DSR and the commands you need to disable them.

10.5 Creating Tables of Contents

10.5.1 How to Create a Table of Contents

To create a table of contents, perform the following steps:

Step	Task
1	Generate an intermediate (binary) file. Be sure to specify an .RNO file type. (DSR then produces a file with a .BRN file type, which contains both table of contents and indexing information.)
2	Run the Table of Contents utility. Be sure to specify a .BRN file type. You can add qualifiers to this command line to customize the table of contents. (DSR then produces a file with an .RNT file type.)
3	Process the .RNT file. Be sure to specify an .RNT file type. (DSR then produces a file with an .MEC file type, which contains the table of contents.)

10.5.2 Table of Contents Default Settings

The RUNOFF/CONTENTS command produces a table of contents with the following features:

- Chapter titles and numbers (generated by the .CHAPTER command).
- Section titles and numbers (generated by the .HEADER LEVEL command). By default, DSR allows up to six levels of headers to appear in the table of contents.
- Appendix titles and letters (generated by the .APPENDIX command).

- Chapter-oriented page numbers (1–1, 1–2, 1–3, . . .) for all table of contents entries.
- An output file with the same name as the input file.

10.5.3 Example: Creating a Table of Contents

The following example shows the commands and default output associated with producing a table of contents:

```
$  RUNOFF/INTERMEDIATE FUN.RNO
$  RUNOFF/CONTENTS FUN.BRN
$  RUNOFF FUN.RNT
$  TYPE FUN.MEC

                                CONTENTS


CHAPTER 1       How to Tile a Floor

        1.1     Reading About Tiling . . . . . 1-1
        1.1.1     Tiling for Fun . . . . . 1-2
        1.1.2     Your Home in Tile  . . . . . 1-3
        1.1.3     Changing a Room with Tile  . . . . . 1-3
        1.2     Buying the Tile  . . . . . 1-5
        1.2.1     Researching Tiles Produced Abroad. . . 1-5
        1.2.2     Coordinating Colors  . . . . . 1-6
        1.2.3     Tile Textures  . . . . . 1-6
        1.2.4     Types of Tiles . . . . . 1-7
        1.2.4.1   Ceramic  . . . . . 1-7
        1.2.4.2   Clay . . . . . 1-7
        1.3     Tools for Tiles  . . . . . 1-8
        1.3.1     Renting a Cutter . . . . . 1-8
        1.3.2     Buying or Renting Crimpers . . . . . 1-9
        1.4     Accompanying Materials . . . . . 1-9
        1.4.1     How to Adhere the Tiles  . . . . . 1-9
        1.4.2     Grout  . . . . . 1-10
CHAPTER 2       How to Cedar a Ceiling

        2.1     Getting Started  . . . . . 2-1
        2.1.1     Various Surfaces . . . . . 2-2
```

10.5.4 DSR Qualifiers for Tailoring a Table of Contents

To tailor the DSR Table of Contents utility to meet your own needs, use these qualifiers:

Qualifier	Result
/BOLD	Enables bolding of chapter and header titles in the table of contents.
/DEEPEST_HEADER=n	Displays header levels up to and including level *n*.
/IDENTIFICATION	Displays the current version number of the DSR Table of Contents utility.
/INDENT	Indents each header level after header level 1, two spaces beyond the preceding header level.

Qualifier	Result
/LOG	Reports the name of each input file as it is processed and after it is processed, plus the name of the generated output file.
/OUTPUT=newfile /NOOUTPUT	Specifies the name of the output file produced by DSR. The /NOOUTPUT qualifier causes DSR to process the input file without creating an output file.
/PAGE_ NUMBERS=RUNNING	Uses running page numbers instead of chapter-oriented page numbers for all table of contents entries, whether or not you specified running page numbers in the document.
/REQUIRE=filespec	Allows you to change the heading on the first page of a table of contents.
/NOSECTION_NUMBERS	Suppresses the display of section numbers for all header levels.
/UNDERLINE	Includes underlining specified in chapter and header titles in the table of contents.

10.5.5 Example

The following example shows how to change the display of page numbers from chapter-oriented numbers (1–1, 1–2, 1–3, . . .) to running numbers (1, 2, 3, . . .):

```
$ RUNOFF/CONTENTS/PAGE_NUMBERS=RUNNING FUN.MEC
$ TYPE FUN.MEC

                                CONTENTS


CHAPTER 1       How to Tile a Floor

        1.1     Reading About Tiling . . . . . . . . . . . . 1
        1.1.1     Tiling for Fun . . . . . . . . . . . . . . 2
        1.1.2     Your Home in Tile  . . . . . . . . . . . . 3
        1.1.3     Changing a Room with Tile  . . . . . . . . 3
        1.2     Buying the Tile  . . . . . . . . . . . . . . 5
        1.2.1     Researching Tiles Produced Abroad. . . . . 5
        1.2.2     Coordinating Colors  . . . . . . . . . . . 6
        1.2.3     Tile Textures  . . . . . . . . . . . . . . 6
        1.2.4     Types of Tiles . . . . . . . . . . . . . . 7
        1.2.4.1   Ceramic  . . . . . . . . . . . . . . . . . 7
        1.2.4.2   Clay . . . . . . . . . . . . . . . . . . . 7
        1.3     Tools for Tiles  . . . . . . . . . . . . . . 8
        1.3.1     Renting a Cutter . . . . . . . . . . . . . 8
        1.3.2     Buying or Renting Crimpers . . . . . . . . 9
        1.4     Accompanying Materials . . . . . . . . . . . 9
        1.4.1     How to Adhere the Tiles  . . . . . . . . . 9
        1.4.2     Grout  . . . . . . . . . . . . . . . . . . 10

CHAPTER 2       How to Cedar a Ceiling

        2.1     Getting Started  . . . . . . . . . . . . . . 1
        2.1.1     Various Surfaces . . . . . . . . . . . . . 2
```

10.6 Creating Indexes

10.6.1 Index Entry Formats

To create an index, enter .INDEX and .ENTRY commands throughout your file. The .INDEX flag will create an index entry with an associated page number. The .ENTRY command will create an index entry without a page number and is usually used for ‘‘See’’ and ‘‘See also’’ entries. The format for an index entry is as follows:

.INDEX topic> subtopic> subtopic

.ENTRY topic [>subtopic >subtopic]

10.6.2 Example

If you want the word ‘‘Chopin’’ with the subtopic "Frederic" to appear in your index, enter the .INDEX command followed by the words ‘‘Chopin>Frederic’’:

```
The music was soft and romantic, and Marvin knew at once that it
.ENABLE INDEXING
.XLOWER
.INDEX Chopin>Frederic
was a waltz by Frederic Chopin that held his attention.
```

In this example, the .ENABLE INDEXING flag enables the operation of the other index commands (.XLOWER and .INDEX). The .XLOWER flag determines that the case of all index entries will be exactly as entered (as opposed to the .XUPPER flag, which automatically capitalizes the first character of every entry and drops everything else in the entry lowercase).

10.6.3 Producing an Index

After you enter the index commands in your file, perform the following procedure:

Step	Task
1	Generate an intermediate (binary) file. Be sure to specify an .RNO file type. (DSR then produces a file with the file type .BRN.)
2	Run the Indexing utility. Be sure to specify a .BRN file type. You can add qualifiers to this command line to customize the Indexing utility. (DSR then produces a file with the file type .RNX.)
3	Process the .RNX file. Be sure to specify an .RNX file type. (DSR then produces a file with the file type .MEX, which contains an index.)

10.6.4 Default Index Settings

The RUNOFF/INDEX command produces an index with the following features:

- Fifty-five lines per page, including the top and bottom header areas

- Chapter-oriented page numbers for index entries
- Consecutive page numbers merged into ranges
- An output file with the same file name as the input file

10.6.5 Example: Producing an Index

The following example shows an index produced by DSR default values:

```
$ RUNOFF/INTERMEDIATE FUN.RNO
$ RUNOFF/INDEX FUN.BRN
$ RUNOFF FUN.RNX
$ TYPE FUN.MEX

                                                    Page Index-1

                              INDEX

Amadeus                              Liszt, Franz, 3-2, 4-11
  See Mozart, Wolfgang Amadeus
                                     Mozart, Wolfgang Amadeus, 3-5,
Bach, Carl Phillip Emanuel, 1-2          4-14
    to 1-4, 4-9
Bach, Johann Sebastian, 1-1, 3-2,    Prokofiev, Sergei, 4-5, 4-15
    4-9, 4-12
Baroque composer                     Rachmaninoff, Sergei, 3-3 to 3-6,
  See Bach, Johann Sebastian             4-13
Bartok, Bela, 2-1, 3-4, 4-10,        Rite of Spring
    4-13                               See Stravinsky, Igor
Britten, Benjamin, 4-3, 4-14
                                     Satie, Erik, 2-2, 4-10
Ceremony of Carols                   Stravinsky, Igor, 4-7, 4-15
  See Britten, Benjamin              Syrinx, 4-8, 4-17
Chopin, Frederic, 4-1 to 4-4
    4-14                             Velvet Gentleman
                                       See Satie, Erik
Debussy, Claude, 3-3, 4-13
                                     Waltz
French composer                        See Chopin, Frederick
  See Debussy, Claude

Hindemith, Paul, 4-5 to 4-7, 4-15
```

10.6.6 DSR Qualifiers for Tailoring an Index

To tailor the DSR Indexing utility to meet your own needs, use the following qualifiers:

Qualifier	Result
/IDENTIFICATION	Displays the current version number of the DSR Indexing utility.
/LINES_PER_PAGE=n	Determines the number of lines of index entries on each page. The number *n* does not include the number of lines required for running heads and feet.

Qualifier	Result
/LOG	Reports the name of each input file as it is processed and after it is processed, plus the name of the generated output file.
/OUTPUT=newfile /NOOUTPUT	Specifies the name of the output file produced by DSR. The /NOOUTPUT qualifier causes DSR to process your input file without creating an output file.
/PAGE_NUMBERS=RUNNING	Uses running page numbers instead of chapter-oriented page numbers for all index entries, whether or not you specified running page numbers in the document.
/REQUIRE=filespec	Allows you to change the heading on the first page of an index.
/RESERVE=n	Allows you to reserve *n* number of lines on the top of the first page of the index.

10.6.7 Example: Tailoring an Index

In the following example, DSR displays index pages that are 15 lines long:

```
$ RUNOFF/INDEX/LINES_PER_PAGE=15 FUN.MEX
$ TYPE FUN.MEX
                                                        Page Index-1

                                INDEX

Amadeus                              Britten, Benjamin, 4-3, 4-14
  See Mozart, Wolfgang Amadeus
                                     Ceremony of Carols
Bach, Carl Phillip Emanuel, 1-2        See Britten, Benjamin
    to 1-4, 4-9                      Chopin, Frederic, 4-1 to 4-4,
Bach, Johann Sebastian, 1-1, 3-2,        4-14
    4-9, 4-12
Baroque composer                     Debussy, Claude, 3-3, 4-13
  See Bach, Johann Sebastian
Bartok, Bela, 2-1, 3-4, 4-10,        French composer
    4-13                               See Debussy, Claude

                                                        Page Index-2
```

Hindemith, Paul, 4-5 to 4-7, 4-15
Liszt, Franz, 3-2, 4-11
Mozart, Wolfgang Amadeus, 3-5, 4-14
Prokofiev, Sergei, 4-5, 4-15
Rachmaninoff, Sergei, 3-3 to 3-5, 4-13
Rite of Spring
  See Stravinsky, Igor
Satie, Erik, 2-2, 4-10
Stravinsky, Igor, 4-7, 4-15
Syrinx, 4-8, 4-17
Velvet Gentleman
  See Satie, Erik
Waltz
  See Chopin, Frederic

10.7 Summary of DSR Commands

10.7.1 Page Size and Running Heads

The following table lists all DSR commands used for modifying page size and running heads:

Command		Description
.AUTOSUBTITLE .NO AUTOSUBTITLE	.AST .NAST	Enable and disable use of section (.HEADER LEVEL) titles for running-head subtitles.
.DATE .NO DATE	.D .ND	Control whether or not the current date appears in running heads. Requires use of the .SUBTITLE command.
.FIRST TITLE	.FT	Allows running-head information to appear on the first page of a document with no chapters. (See also .HEADERS ON, .LAYOUT, .TITLE, .SUBTITLE, and .AUTOSUBTITLE.)
.HEADERS ON .NO HEADERS	.HD .NHD	Restore and cancel the capability of having one or two lines of information, called running heads, at the top of a page. Running heads indicate the content of the page and the page number.
.HEADERS UPPER .HEADERS LOWER .HEADERS MIXED	.HD UPPER .HD LOWER .HD MIXED	Specify the case of the word ‘‘page’’ that precedes the page number. The commands produce, respectively, PAGE, page, and Page. In an index, these commands also affect the word ‘‘index’’ that is part of the page number (for example, Page Index-3). The command normally takes effect on the next page.
.LAYOUT	.LO	Rearranges running-head and running-foot information on pages.
.PAGE SIZE	.PS	Sets the page ‘‘frame’’ by specifying the page length (the maximum number of lines of text on a page) and the page width for the running heads.

Command		Description
.SUBTITLE .NO SUBTITLE	.ST .NST	Allow you to specify a subtitle for a running head (see .HEADERS ON).
.TITLE	.T	Allows you to specify a title for a running head (see .HEADERS ON). This title normally appears at the top of every page but the first, at the leftmost position on the line (character position 0), regardless of the .LEFT MARGIN setting. (See also .FIRST TITLE, .SUBTITLE, and .LAYOUT.)

10.7.2 Paging and Page-Number Control

The following table lists all DSR commands used for controlling paging and page numbers:

Command		Description
.DISPLAY NUMBER	.DNM	Allows you to specify the form that sequential numbering (or lettering) of pages will take.
.NO NUMBER .NUMBER PAGE	.NNM .NMPG	Suspend and resume normal page numbering. The .NUMBER PAGE command keeps track of the numbering while the .NO NUMBER command is in effect; or, it allows you to specify the beginning of a new number sequence by specifying a number for the next page. (See also .NUMBER RUNNING, .DISPLAY NUMBER, .NO PAGING, and .HEADERS ON.)
.NUMBER RUNNING	.NMR	Allows you to specify the beginning of a new sequence of running page numbers. This command affects page numbers only if you have entered a .LAYOUT command with an *n1* value of 3. (See .LAYOUT, .HEADERS ON, and .NO NUMBER.)
.PAGING .NO PAGING	.PS .NPA	Enable and disable paging, which splits a document into numbered pages and reserves space for running heads. This command has no effect on help files (files with the file type .RNH).

10.7.3 Subpaging

The following table lists all DSR commands used for subpaging:

Command		Description
.DISPLAY SUBPAGE	.DSP	Allows you to specify the form that sequential lettering (or numbering) of subpage characters will take.

Command		Description
.NUMBER SUBPAGE	.NMSPSG	Allows you to specify the beginning of a new sequence of subpage numbers, for example, 1-16A, 1-16B, 1-16C, and so on. This command affects only the letters that the .SUBPAGE command appends to the normally numeric page number. .NUMBER SUBPAGE takes effect on the next page. (See also .SUBPAGE and .DISPLAY SUBPAGE.)
.SUBPAGE .END SUBPAGE	.SPG .ES	Begin and end a new page and a new format for page numbering. (See also .NUMBER SUBPAGE, .DISPLAY SUBPAGE, .HEADERS ON, .LAYOUT, and .PAGE.)

10.7.4 Margin Settings

The following table lists all DSR commands used for setting margins:

Command		Description
.LEFT MARGIN	.LM	Sets the left margin to the specified position.
.RIGHT MARGIN	.RM	Sets the right margin to the specified position. This is the position to which a line of text normally extends. If .JUSTIFY is in effect, the .RIGHT MARGIN value is the position against which text is justified. If .NO JUSTIFY is in effect, the .RIGHT MARGIN value specifies the maximum number of characters on any text line.

10.7.5 Filling and Justifying Text

The following table lists all DSR commands used for filling and justifying text:

Command		Description
.AUTOJUSTIFY .NO AUTOJUSTIFY	.AJ .NAJ	Justifies and fills text automatically within the context of an appendix, chapter, section, or note. The .NO AUTOJUSTIFY command disables automatic justification. If you disable justification and filling with the .NO JUSTIFY command, the settings for .[NO]FILL and .[NO]JUSTIFY will remain in effect.
.FILL .NO FILL	.F .NF	Fills each line with words until the addition of one more word would exceed the right margin. The .NO FILL command suspends both line filling and justification.
.JUSTIFY .NO JUSTIFY	.J .NJ	Inserts exactly enough space between words so that the last character reaches the right margin. The .NO JUSTIFY command disables justification.

10.7.6 Vertical Spacing

The following table lists all DSR commands used for controlling vertical spacing:

Command		Description
.BLANK	.B	Inserts exactly the number of blank lines that you specify (for example, .B2, .B3).
.BREAK	.BR	Ends the current line immediately, without filling or justifying the text.
.KEEP .NO KEEP	.K .NK	Allow you to keep or not keep blank lines in the output that are present in the input file when .NO FILL is in effect. Normally multiple blank lines in the input file are discarded. (See also .LITERAL.)
.SKIP	.S	Inserts a multiple of the number of blank lines that has been specified by the .SPACING command.
.SPACING	.SP	Changes the amount of spacing between lines of text.
.PAGE	.PG	Starts a new page.
.TEST PAGE	.TP	Allows you to keep a specified amount of text entirely on a single page. If there is not enough room on the current page to accommodate that amount, DSR ends the current page and puts the entire text on the next page.

10.7.7 Horizontal Spacing

The following table lists all DSR commands used for controlling horizontal spacing:

Command		Description
.CENTER .CENTER	.C	Centers a single line of text around a character position on a line.
.INDENT	.I	Causes the first line of text following it to begin at a position relative to the left margin.
.NO PERIOD .PERIOD	.NPR .PR	Cancel and restore the routine insertion of an extra space after any of the following punctuation marks: period (.), colon (:), question mark (?), and exclamation point (!).
.RIGHT	.R	Positions a single line of text relative to the right margin. (See also .CENTER.)
.TAB STOPS	.TS	Changes the current positions of tab stops. Each tab character in the input file advances the print carriage to the right to the next tab stop.

10.7.8 Paragraph Formatting

The following table lists all DSR commands used for formatting paragraphs:

Command		Description
.AUTOPARAGRAPH .NO AUTOPARAGRAPH	.AP .NAP	Enable and disable starting a new paragraph each time a line starts with a space, a tab, or a blank line. Cancels .AUTOTABLE.
.AUTOTABLE .NO AUTOTABLE	.AT .NAT	Enable and disable starting a new paragraph each time a line does not start with a space or a tab. Cancels .AUTOPARAGRAPH.
.PARAGRAPH	.P	Controls spacing and page placement associated with the creation of paragraphs. (See also .SET PARAGRAPH.)
.SET PARAGRAPH	.SPR	Allows you to set values for .PARAGRAPH without entering .PARAGRAPH (for example, when you are using .AUTOPARAGRAPH).

10.7.9 Text Emphasis

The following table lists all DSR commands used for emphasizing text:

Command		Description
.ENABLE BAR .DISABLE BAR	.EBB .DBB	Enable and disable the use of change bars, vertical bars (|) inserted to indicate where changes in text have occurred since the previous edition of a document.
.BEGIN BAR .END BAR	.BB .EB	Determine where DSR starts and stops inserting change bars at the beginning of lines.
.ENABLE BOLDING .DISABLE BOLDING	.EBO .DBO	Enable and disable use of the Bold flag (*) to indicate bolding, if the .FLAGS BOLD flag is enabled.
.ENABLE HYPHENATION .DISABLE HYPHENATION	.EHY .DHY	Enable and disable use of the Hyphenate flag (=) to indicate hyphenation, if the .FLAGS HYPHENATE flag is enabled.
.ENABLE OVERSTRIKING .DISABLE OVERSTRIKING	.EOV .DOV	Enable and disable use of the Overstrike flag (%) to create special characters that are not available on the terminal by overstriking any printing character with another. Recognition of the .FLAGS OVERSTRIKE flag must be enabled.
.ENABLE UNDERLINING .DISABLE UNDERLINING	.EUN .DUL	Enable and disable use of the Underline flag (&) to underline text, if the .FLAGS UNDERLINE flag is enabled.

10.7.10 Figures

The following table lists all DSR commands used for formatting figures:

Command		Description
.FIGURE	.FG	Leaves room on a page for you to insert a figure later. If there is not enough room on the current page, DSR ends the page immediately and then puts the blank lines at the top of the next page.
.FIGURE DEFERRED	.FGD	Leaves room on a page for you to insert a figure later. If there is not enough room on the current page, .FIGURE DEFERRED first adds enough text to complete the page and then puts the required number of blank lines at the top of the next page.
.LITERAL .END LITERAL	.LT .EL	Allows you to have your text formatted exactly as you have typed it. DSR commands and flags are not recognized and are treated as ordinary text. Tab stops set prior to the .LITERAL command, however, remain in effect (see .TAB STOPS).

10.7.11 Lists

The following table lists all DSR commands used for formatting lists:

Command		Description
.DISPLAY ELEMENTS	.DLE	Allows you to specify the form that sequential numbering or lettering of items in a list will take.
.LIST .END LIST	.LS .ELS	Specifies the beginning of a list by resetting the left margin farther to the right, by setting a .SKIP command value to take effect before each item in the list, and by executing the .TEST PAGE command. The .END LIST command ends a list, restoring any fill, justify, case, margin, or spacing settings that were in effect before you entered the most recent .LIST command.
.LIST ELEMENT	.LE	Specifies the beginning of each item in a list.
.NUMBER LIST	.NMLS	Allows you to specify, anywhere in a list, the number with which a sequence of items in a list will begin. Enter this command just before the .LIST ELEMENT command that you want to affect. Subsequent list elements will each have a number that is one greater than the number for the preceding .LIST ELEMENT command. (See also .DISPLAY ELEMENTS, with which you can specify the form the number will take.)

10.7.12 Notes and Footnotes

The following table lists all DSR commands used for creating notes and footnotes:

Command		Description
.FOOTNOTE .END FOOTNOTE	.FN .EFN	Places text at the bottom of the current page. If there is not enough space on the current page for the entire footnote, DSR places the entire note at the bottom of the next page. The .END FOOTNOTE command ends the footnote and restores any case, fill, justify, spacing, or margin settings that you might have changed within the footnote.
.NOTE .END NOTE	.NT .EN	Highlights a portion of text by narrowing the margin settings, centering the text on the page, and printing a title centered over the text. The .END NOTE command restores the fill, justify, case, margin, and spacing settings that were in effect just before you entered the .NOTE.

10.7.13 Chapters and Appendixes

The following table lists all DSR commands used for formatting chapters and appendixes:

Command		Description
.APPENDIX	.AX	Specifies the beginning of an appendix, assigns an identifying letter to it, and allows you to supply a title.
.CHAPTER	.CH	Specifies the beginning of a chapter, assigns an identifying number to it, and allows you to supply a title.
.DISPLAY APPENDIX	.DAX	Allows you to specify the form that the lettering (or numbering) of appendixes will take. The form you specify appears in the title, the page numbers, and the first character of header-level numbers throughout the appendix.
.DISPLAY CHAPTER	.DCH	Allows you to specify the form that the numbering (or lettering) of chapters will take. The form you specify appears in the title, the page numbers, and the first character of header-level numbers throughout the chapter.
.NUMBER APPENDIX	.NMAX	Allows you to specify an identifying letter with which a sequence of appendixes will begin. The next .APPENDIX command starts the sequence. Subsequent .APPENDIX commands cause appendixes to be lettered in alphabetic order. (See also .DISPLAY APPENDIX.)

Command		Description
.NUMBER CHAPTER	.NMCH	Allows you to specify the number with which a sequence of chapters will begin. The next .CHAPTER command starts the sequence. Subsequent .CHAPTER commands will cause each chapter to be numbered one higher than the previous chapter. (See also .DISPLAY CHAPTER.)

10.7.14 Sections

The following table lists all DSR commands used for formatting sections:

Command		Description
.DISPLAY LEVELS	.DHL	Allows you to specify the form that sequential numbering (or lettering) of section headers will take.
.HEADER LEVEL	.HL	Allows you to specify both a section number and a section title. Successive .HEADER LEVEL commands of the same value cause the section numbers to increase sequentially (all .HEADER LEVEL 1 commands, for example).
.NUMBER LEVEL	.NMLV	Allows you to specify the beginning number of a sequence of headers. (See also .STYLE HEADERS and .DISPLAY LEVELS.)
.SET LEVEL	.SL	Allows you to preset the level of the next section head without entering a .HEADER LEVEL command (see .HEADER LEVEL).
.STYLE HEADERS	.STHL	Changes the formats of the various levels of section headers (.HEADER LEVEL n).

10.7.15 Indexes

The following table lists all DSR commands used for formatting indexes:

Command		Description
.ENABLE INDEXING .DISABLE INDEXING	.EIX .DIX	Enable and disable the operation of the indexing commands (.INDEX and .ENTRY) and the Index flag (>).
.ENTRY	.Y	Creates an index entry without a page number reference. It is usually used for ''See'' or ''See also'' index entries.
.FLAGS INDEX .NO FLAGS INDEX	.FL INDEX .NFL INDEX	Enable and disable recognition of the Index flag character (>).
.FLAGS SUBINDEX .NO FLAGS SUBINDEX	.FL SUBINDEX .NFL SUBINDEX	Enable and disable recognition of the Subindex flag (>). The .FLAGS SUBINDEX command can change the Subindex flag to another character. The .NO FLAGS SUBINDEX can include a right angle bracket (>) as part of your indexed text, instead of causing subindexing.
.INDEX	.X	Creates an index entry with a page number reference.
.XLOWER .XUPPER	.XL .XU	Determine whether you control the case of index entries specified by the .INDEX and the .ENTRY commands or by the Index flag (>). When you enter the .XLOWER command, the case of the index entries matches exactly the case that you enter when you make the index entry. When you enter the .XUPPER command, DSR capitalizes the first character of every index entry and drops everything else in the entry to lowercase.

10.7.16 Tables of Contents

The following table lists all DSR commands used for formatting tables of contents:

Command		Description
.ENABLE TOC .DISABLE TOC	.ETC .DTC	Enable and disable use of information collected by DSR to create a table of contents.
.SEND TOC	.STC	Allows you to control the appearance of the table of contents (.RNT) file by inserting DSR commands, DSR flags, and text.

10.7.17 Flag Recognition Commands

The following table lists all DSR commands used for flag recognition:

Command		Description
.FLAGS ACCEPT .NO FLAGS ACCEPT	.FL ACCEPT .NFL ACCEPT	Control recognition of the Accept flag character (_).
.FLAGS ALL .NO FLAGS ALL	.FL .NFL	Serve as master switches for all other flag (.FLAG and .NO FLAG) settings, except the .FLAGS COMMENT, .NO FLAGS COMMENT, .FLAGS CONTROL, and .NO FLAGS CONTROL commands. The .FLAGS ALL and .NO FLAGS ALL commands enable and disable recognition of all flags without disturbing other flag command settings. (An analogy for flag recognition is turning on a master switch [entering .FLAGS ALL] — those lights with switches in the ON position will go on and those with switches in the OFF position will not go on.) See also .ENABLE BOLDING and .DISABLE BOLDING, .HYPHENATION, .INDEXING, .OVERSTRIKING, and .UNDERLINING commands.
.FLAGS BOLD .NO FLAGS BOLD	.FL BOLD .NFL BOLD	Enable and disable recognition of the Bold flag character (*).
.FLAGS BREAK .NO FLAGS BREAK	.FL BREAK .NFL BREAK	Enable and disable recognition of the Break flag character (|), which specifies the place at which a new page should begin.
.FLAGS CAPITALIZE .NO FLAGS CAPITALIZE	.FL CAPITALIZE .NFL CAPITALIZE	Enable and disable recognition of the Capitalize flag character (<).
.FLAGS COMMENT .NO FLAGS COMMENT	.FL COMMENT .NFL COMMENT	Enable and disable recognition of the Comment flag character (!).
.FLAGS CONTROL .NO FLAGS CONTROL	.FL CONTROL .NFL CONTROL	Control recognition of the Control flag character (the period that begins a DSR command). The .FLAGS CONTROL command changes the character that precedes the commands from a period to a character of your choice. The .NO FLAGS CONTROL command disables recognition of the Control flag character.

Command		Description
.FLAGS HYPHENATE .NO FLAGS HYPHENATE	.FL HYPHENATE .NFL HYPHENATE	Enable and disable recognition of the Hyphenate flag character (=).
.FLAGS LOWERCASE .NO FLAGS LOWERCASE	.FL LOWERCASE .NFL LOWERCASE	Enable and disable recognition of the Lowercase flag character (\).
.FLAGS OVERSTRIKE .NO FLAGS OVERSTRIKE	.FL OVERSTRIKE .NFL OVERSTRIKE	Enable and disable recognition of the Overstrike flag character (%).
.FLAGS PERIOD .NO FLAGS PERIOD	.FL PERIOD .NFL PERIOD	Enable and disable recognition of the Period flag character (+).
.FLAGS SPACE .NO FLAGS SPACE	.FL SPACE .NFL SPACE	Enable and disable recognition of the Space flag character (#).
.FLAGS SUBSTITUTE .NO FLAGS SUBSTITUTE	.FL SUBSTITUTE .NFL SUBSTITUTE	Enable and disable recognition of the Substitute flag character ($). Note that you must use a pair of Substitute flag characters to make the substitution occur.
.FLAGS UNDERLINE .NO FLAGS UNDERLINE	.FL UNDERLINE .NFL UNDERLINE	Enable and disable recognition of the Underline flag character (&).
.FLAGS UPPERCASE .NO FLAGS UPPERCASE	.FL UPPERCASE .NFL UPPERCASE	Enable and disable recognition of the Uppercase flag (^).

10.7.18 Other DSR Commands

The following table lists other miscellaneous DSR commands:

Command		Description
.CONTROL CHARACTERS .NO CONTROL CHARACTERS	.CC .NCC	Enable and disable use of control characters as normal text in your input file.
.IF .IFNOT .ELSE .ENDIF	– – .EI .IN	Cause portions of a DSR file to be processed or not processed, according to conditions that you specify.
.NO SPACE	.NSP	Prevents the insertion of the end-of-line space for one line of text only, causing the characters at the end of one line and the beginning of the next to be adjacent.
.REPEAT	.RPT	Allows you to specify up to 150 characters to be printed a specified number of times, either horizontally or vertically.

Command		Description
.REQUIRE	.REQ	Allows you to process several DSR files at the same time and merge them in an output file.
.SAVE .RESTORE	.SA .RE	Maintain the current RUNOFF formatting context of a document, including DSR defaults as well as DSR commands and flags.
.SET DATE .SET TIME	.SDT .STM	Specify a date and time to be inserted in your file when you use the Substitute flag pair, $$, with any of the appropriate date or time parameters. The .SET DATE command also sets the date for the .DATE command, which causes the date to appear in running heads.
.VARIABLE	.VR	Allows you to specify a character that corresponds to the name you have given the commands and text in an .IF (or .IFNOT) block. This identifying character is placed in the left margin when you process your file with the /DEBUG or /DEBUG=CONDITIONALS command line qualifier.

11

Sort/Merge Utility: Sorting and Merging Files

11.1 Overview

This chapter describes how to use the OpenVMS Sort/Merge utility (SORT/MERGE). The Sort/Merge utility performs two operations:

- Sorts records from one or more input files according to the fields you select and generates one reordered output file
- Merges up to 10† input files that have been sorted previously according to the same key fields and generates one output file.

11.1.1 Using High-Performance Sort/Merge

Alpha

On Alpha systems, you can also choose the high-performance Sort/Merge utility. This utility takes advantage of the Alpha architecture to provide better performance for most Sort and Merge operations.

The high-performance Sort/Merge utility uses the same command line interface as SORT/MERGE. Any differences between the high-performance Sort/Merge utility and SORT/MERGE are highlighted by footnotes within the text of this chapter.

Use the SORTSHR logical to select the high-performance Sort /Merge utility. Define SORTSHR to point to the high-performance sort executable in SYS$LIBRARY as follows:

```
$ define sortshr sys$library:hypersort.exe
```

To return to SORT/MERGE, deassign SORTSHR. The SORT /MERGE utility is the default if SORTSHR is not defined.♦

11.1.2 About This Chapter

This chapter describes:

- Sorting files
- Specifying collating sequences
- Running Sort as a batch job
- Merging files
- Entering records from a terminal
- Using a Sort/Merge specification file
- Optimizing a Sort or Merge operation
- Summary of Sort/Merge qualifiers

† The high-performance Sort/Merge utility supports up to 12 input files.

11.1.3 References

- For additional information on commands used in this chapter, refer to the *OpenVMS DCL Dictionary*.
- For information on how a system manager can improve efficiency when using the Sort/Merge utility, see the *OpenVMS System Manager's Manual*.

11.2 Sorting Files

This section describes how to sort records from one or more input files.

11.2.1 SORT Command

To sort files, use the DCL command SORT. Specify the names of the files to be sorted, separated by commas, followed by the name of the ordered output file to be created.

11.2.2 Sort Keys

Optionally, you can specify a key for each field on which you want to sort. Each key includes the following information:

- Starting position of the key field in a record (required)
- Size of the key (required)
- Data type of the key
- Order in which the records are sorted
- Priority of the key

If you do not specify any keys, Sort assumes there is only one key and that this key field:

- Begins in the first position of a record
- Includes the entire record
- Contains character data
- Is sorted in ascending order

Examples

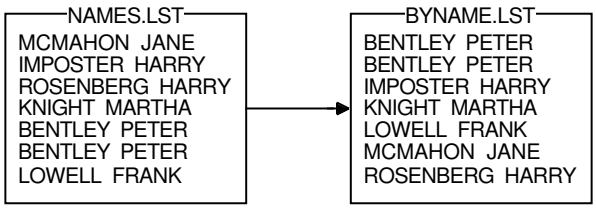
The following two examples use the default key.

1. In this example, the file NAMES.LST is sorted in ascending order:

   ```
   $ SORT NAMES.LST BYNAME.LST
   ```

   This command creates the ordered output file BYNAME.LST, as shown in Figure 11–1:

Figure 11–1 List Sorted in Ascending Order



2. In this example, the files NAMES.LST and NAMES2.LST are sorted into the ordered output file BYNAME.LST. Sort treats the files as if they were one large file:

```
$ SORT NAMES.LST,NAMES2.LST  BYNAME.LST
```

See Section 11.9 for a complete list of SORT qualifiers.

11.2.3 Defining a Key

Use the /KEY qualifier to define a key. When specifying multiple keys, use a separate /KEY qualifier for each key.

Table 11–1 describes the five elements that comprise a key.

Table 11–1 /KEY Qualifier Values

Key Element	Value	Description
Key position	POSITION:*n*	The position of the first byte of the key field within the record. The first byte in a record is position 1. POSITION:*n* is required.
Key size	SIZE:*n*	The length of the key field. SIZE:*n* is required except for floating point data. The data type you specify for the key determines what values are acceptable when specifying size. The following table lists the possible values for each type of data and the units used to specify the size of the key.

Data	Valid Range	Units
Character	1 through 32,767	Characters
Binary	1, 2, 4, 8, or 16†	Bytes
Decimal	1 through 31	Digits
Floating-point	No value is necessary.	

For decimal data, if the decimal sign is stored in a separate byte, that byte is not counted toward the size of the data.

If you specify a key that extends beyond the end of a record, Sort treats the missing characters as null characters.

†For the high-performance Sort/Merge utility, the size of a binary data type key must be 1, 2, 4, or 8 bytes. Support of a 16-byte binary key is deferred to a future OpenVMS Alpha release.

(continued on next page)

Table 11–1 (Cont.) /KEY Qualifier Values

Key Element	Value	Description
Data type	CHARACTER	Character data. CHARACTER is the default data type.
	BINARY	Binary data.
		SIGNED — Signed binary or decimal data. SIGNED is the default for binary and decimal data.
		UNSIGNED — Unsigned binary or decimal data.
	F_FLOATING	F_FLOATING format data.
	D_FLOATING	D_FLOATING format data.
	G_FLOATING	G_FLOATING format data.
	H_FLOATING‡	On VAX systems, H_FLOATING format data.
	S_FLOATING	On Alpha systems, IEEE S_FLOATING format data.
	T_FLOATING	On Alpha systems, IEEE T_FLOATING format data.
	DECIMAL	Decimal data.
		TRAILING_SIGN — Trailing sign decimal data. TRAILING_SIGN is the default for decimal data.
		LEADING_SIGN — Leading sign decimal data. The leading sign must be in the first position of the field and the field must be left zero padded.
		OVERPUNCHED_SIGN — Overpunched decimal data. OVERPUNCHED_SIGN is the default for decimal data.
		SEPARATE_SIGN — Separate sign decimal data.
	ZONED‡	Zoned decimal data.
	PACKED_DECIMAL	Packed decimal data.
Sort order	ASCENDING	Orders the sorting operation in ascending alphabetical or numerical order. ASCENDING is the default order.
	DESCENDING	Orders the sorting operation in descending alphabetical or numerical order.
Key priority	NUMBER:*n*	Specifies the order of priority of each key if you do not list multiple keys in the order of their priority. A value of 1 to 255 can be specified.

‡Not currently supported by the high-performance Sort/Merge utility.

If the data in the key fields is not character data, you must specify the data type. The following data types are recognized by the Sort/Merge utility:

BINARY, [SIGNED]
BINARY, UNSIGNED
CHARACTER
DECIMAL, LEADING_SIGN, SEPARATE_SIGN [SIGNED]

DECIMAL, LEADING_SIGN, [OVERPUNCHED_SIGN, SIGNED]
DECIMAL [,SIGNED, TRAILING_SIGN, OVERPUNCHED_SIGN]
DECIMAL, [TRAILING SIGN], SEPARATE_SIGN, [SIGNED]
DECIMAL, UNSIGNED
D_FLOATING
F_FLOATING
G_FLOATING
H_FLOATING
S_FLOATING, IEEE (Alpha systems only)
T_FLOATING, IEEE (Alpha systems only)
PACKED_DECIMAL
ZONED

The items in brackets are defaults and need not be specified.

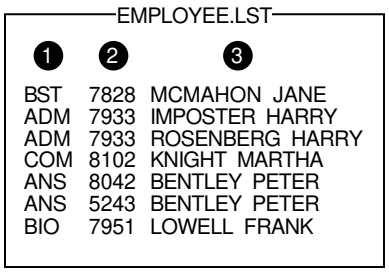
Note

For decimal string data, the Sort/Merge utility reports an invalid digit in the input string differently for VAX and Alpha systems. On VAX systems, you receive a message that the invalid digit (or reserved operand) is converted to a valid decimal string for comparison purposes. On Alpha systems, Sort/Merge performs the same conversion but does not display a message. In both cases, the data from the input file is written to the output file without change.

Examples

In Figure 11–2, each record in the file EMPLOYEE.LST consists of three fields: a department name, an account number, and a customer name:

Figure 11–2 Record Fields in a List



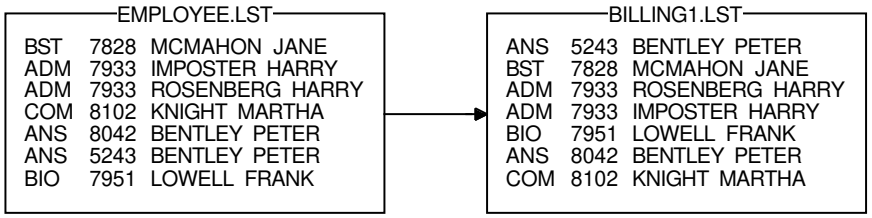
ZK–5056A–GE

1. In the following example, EMPLOYEE.LST is sorted by account number, using the /KEY qualifier to describe the account number field:

```
$ SORT/KEY=(POSITION:5,SIZE:4,DECIMAL)  EMPLOYEE.LST BILLING1.LST
```

This command specifies that the key field (the account number) starts in position 5, is 4 characters long, contains decimal data, and should be sorted in ascending order (the default). Figure 11–3 shows the results of this Sort operation.

Figure 11–3 Sorting by Key Field



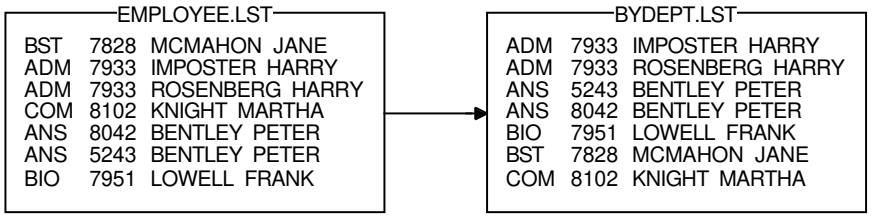
ZK–5058A–GE

2. The following example shows how to sort the file EMPLOYEE.LST without specifying a key field:

```
$  SORT EMPLOYEE.LST BYDEPT.LST
```

Because no key is specified, Sort assumes the default characteristics. Figure 11–4 shows the result of this Sort operation.

Figure 11–4 Sorting with Default Key Records



ZK–5057A–GE

Sort treats each record in EMPLOYEE.LST as one key of character data. In this example, each record includes a department name, an account number, and a customer name. If Sort finds a duplicate department name, it sorts the names by account number. If it then finds a duplicate account number, it sorts by customer name. Note that the account number is part of the record. Unless you specify otherwise, it is treated as character data.

11.2.4 Multiple Key Fields

You can sort with more than one key (up to a limit of 255 keys). You can specify multiple keys in order of their priority with the primary key first, the secondary key next, and so on. Alternately, you can specify a key's priority using NUMBER:*n*. Each key can be ascending or descending.

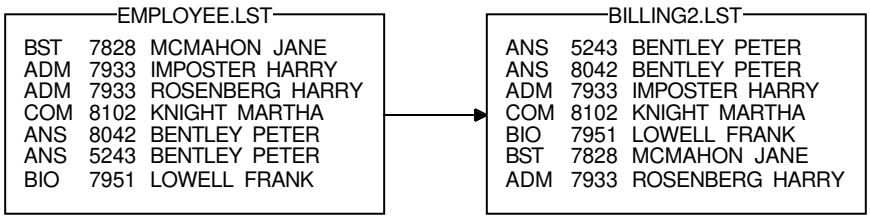
Examples

1. In the following example, the file EMPLOYEE.LST is sorted by the customer name key first and then (where there are identical names), by the account number:

```
$  SORT /KEY=(POSITION:10,SIZE:15,CHARACTER) -
_$ /KEY=(POSITION:5,SIZE:4,DECIMAL) EMPLOYEE.LST BILLING2.LST
```

Figure 11–5 shows the results of this Sort operation.

Figure 11–5 Sorting with Multiple Key Fields



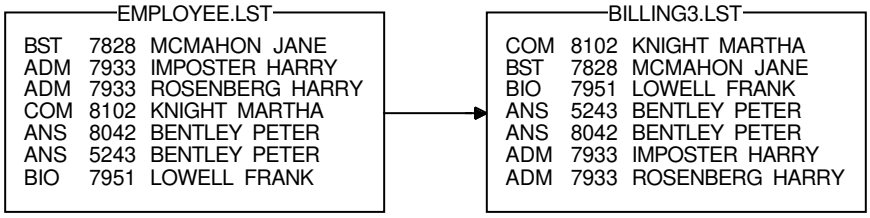
ZK–5059A–GE

2. In the following example, records are sorted first by the department name in descending order, then by the customer name in ascending order:

```
$ SORT/KEY=(POSITION:1,SIZE:3,DESCENDING) -
_$ /KEY=(POSITION:10,SIZE:15) -
_$ EMPLOYEE.LST BILLING3.LST
```

Figure 11–6 shows the results of this Sort operation.

Figure 11–6 Sorting with Multiple Key Fields (Ascending and Descending Order)



ZK–5060A–GE

11.2.5 Identical Key Fields

By default, Sort/Merge keeps records with identical key fields but does not necessarily maintain the same order in which they appeared in the input file. To control the way in which records with identical keys are sorted, specify one of the following qualifiers:

- /STABLE

  Maintains the input order of records with identical keys. If you use this qualifier when sorting multiple input files, on output, records with equal keys in the first file precede those from the second file and so on.

- /NODUPLICATES

  Retains only one copy of records with identical keys. If you want to specify which duplicate record to keep, invoke Sort at the program level and specify an equal-key routine.

The /STABLE and /NODUPLICATES qualifiers are incompatible. You cannot specify both qualifiers on the same command line.

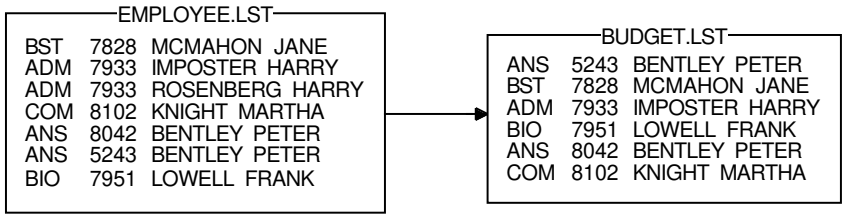
Example

In the following example, records with duplicate account numbers are eliminated from the file EMPLOYEE.LST:

```
$  SORT /KEY=(POSITION:5,SIZE:4)/NODUPLICATES EMPLOYEE.LST BUDGET.LST
```

Figure 11–7 shows the results of this Sort operation.

Figure 11–7 Sorting with Identical Key Fields



ZK–5061A–GE

11.2.6 Noncharacter Data

If you sort records that contain items other than character data, specify the data type of each key. In addition, take care in calculating starting positions and sizes because the items being compared can occupy more than 1 byte.

Example

If you are sorting a file that contains 20 characters followed by 3 floating-point numbers in F_floating format, the positions are as follows:

- The character data occupies positions 1 to 20 (20 characters).
- The first F_floating-point number occupies positions 21 to 24.
- The second F_floating-point number occupies positions 25 to 28.
- The third F_floating-point number occupies positions 29 to 32.

To sort the file by the third floating-point number, specify the key field as follows:

```
$ SORT/KEY=(POSITION:29,F_FLOATING) STATS.RAW STATS.SOR
```

You do not need to specify the size of the floating-point number because it is fixed at four bytes.

11.2.7 Output File Organization

By default, Sort produces an output file with the same file organization as that of the first input file. To specify a different output file organization, include one of the following qualifiers after the output file specification on the Sort command line:

- /FORMAT (record format)

  When you use this output qualifier, you can define the file record format, length, and block size.

- /INDEXED_SEQUENTIAL†

  Using this qualifier, you can define the output to have **indexed sequential file** organization. If you specify indexed sequential as the output file organization, you must also do the following:

  – Before you perform the Sort operation, create an empty file to be used as the output file. Sort requires an output file that already exists and is empty.

  – Include the /OVERLAY qualifier after the name of the output file on the SORT command line. The /OVERLAY qualifier indicates the existing file is to be overlaid with the sorted records of the input file.

- /RELATIVE

  Using this qualifier, you can define the output to have **relative file organization**.

- /SEQUENTIAL

  Using this qualifier, you can define the output to have **sequential file organization**.

Example

In the following example, a sequential file is produced after the indexed sequential file EMPLOYEE.LST is sorted:

```
$ SORT/KEY=(POSITION:10,SIZE:15) -
_$ EMPLOYEE.LST BYNAME.LST/SEQUENTIAL
```

11.2.8 Sorting Process

Sort arranges files using one of the internal processes: record, tag, address, or indexed.‡ The process you specify can affect the efficiency of the Sort operation. Refer to Section 11.8 for information about optimizing a Sort or Merge operation.

The following table describes the four types of process. Use the /PROCESS=*type* qualifier to specify the sort process.

† The high-performance Sort/Merge utility does not support indexed sequential output file organization. Implementation of this feature is deferred to a future OpenVMS Alpha release.

‡ The high-performance Sort/Merge utility supports only the record process. Implementation of tag, address, and index processes is deferred to a future OpenVMS Alpha release.

Sort Process	*type*	Description
Record	RECORD	Keeps records intact while sorting and produces an output file consisting of complete records. Record is the default sorting process.
Tag	TAG	Sorts the key fields only and then rereads the input file to produce an output file of complete records. The net result is the same as for a complete record sort. A tag sort is useful if disk space is low because it typically uses less work file space during the sorting. In most cases, a tag sort is slower than a record sort because it requires extra time to reread the input file.
Address	ADDRESS	Sorts the key fields only and produces an output file that is an index of **record file addresses (RFAs)** in binary format. An address sort is faster than a record sort but you must write a program to associate the record addresses with the records of the input file.
Indexed	INDEX	Sorts the key fields only and produces an output file of keys and RFAs (in binary format). As with an address sort, an index sort is faster than a record sort, but you must write a program to associate the record addresses with the records of the input file.

11.3 Specifying a Collating Sequence

Characters are sorted according to a **collating sequence**. For files that contain character data, you can use the /COLLATING_SEQUENCE=*sequence* qualifier to specify the collating sequence. The following table describes the collating sequence options.

Collating Sequence	*sequence*	Description
ASCII	ASCII	The default collating sequence for character data. The ASCII sequence orders numbers (0 to 9) first, then uppercase letters (A to Z), and then lowercase letters (a to z).
EBCDIC	EBCDIC	Generates an output file that is ordered in EBCDIC sequence. The data remains in the ASCII representation. The EBCDIC sequence orders lowercase letters (a to z) first, then uppercase letters (A to Z), and then numbers (0 to 9).
DEC Multinational character set	MULTINATIONAL	The multinational collating sequence collates characters according to the DEC Multinational character set (refer to Appendix B). In the MULTINATIONAL character sequence, characters are ordered according to the following rules: • All diacritical forms of a character are given the collating value of the character (A', A", A‘ collate as A). • Lowercase characters are given the collating value of their uppercase equivalents (a collates as A, a" collates as A"). • If two strings compare as equal, tie-breaking is performed. The strings are compared to detect differences due to diacritical marks, ignored characters, or characters that collate as equal although they are actually different. If strings still compare as equal, another comparison is done based on the numeric codes of the characters. In this final comparison, lowercase characters are ordered before uppercase.
National character set (NCS)†	Collating sequence name	The named collating sequence must be defined in an NCS library. For more information, see the *OpenVMS National Character Set Utility Manual*. NCS collating sequences are supported only through the command line interface and not through specification files.

†The high-performance Sort/Merge utility does not support the National Character Set (NCS) collating sequences. Support for NCS collating sequences is deferred to a future OpenVMS Alpha release.

User-defined sequence‡	(sequence-string)	Specifies a user-defined collating sequence. User-defined collating sequences are supported only through specification files and not through the command line interface. Define a collating sequence by specifying a string of single or double characters or ranges of single characters. (A double character is any set of two single characters collated as if they were one character. For example, "CH" can be defined to collate as "C".) This string should be enclosed in parentheses. You can also represent characters by their corresponding octal, decimal, or hexadecimal values using the radix operators: %O, %D, %X. You must observe the following rules when defining your collating sequence: • Enclose characters in quotation marks (""). • Separate each character and character range with a comma (,), and enclose the entire list in parentheses. • Give all the characters appearing in the character keys in the Sort or Merge operation a collating value. Any character not given a collating value will be ignored unless the FOLD or MODIFICATION options are specified. • Do not define a character more than once. • Do not specify the null character by using quotation marks (""). Instead, use a radix operator such as %X0. • Specify quotation marks by enclosing them within another set of quotation marks ("" "") or by using a radix operator. The following string defines a collating sequence in which the double character LL collates as a single character between L and M. `("A"-"L","LL","M"-"Z")`

‡The high-performance Sort/Merge utility does not support user-defined collating sequences. Support for user-defined collating sequences is deferred to a future OpenVMS Alpha release.

Note

Exercise caution when using the multinational collating sequence to sort or merge files for further processing. Sequence-checking procedures in most programming languages compare numeric characters. Normal sequence checking does not work because the multinational sequence is based on actual graphic characters, not the codes representing those characters.

11.3.1 Examples

The following examples demonstrate the creation of user-defined collating sequences for use in specification files. See Section 11.7 for information about specification files.

1. `/COLLATING_SEQUENCE=(SEQUENCE=ASCII,IGNORE=("-"," "))`

   This /COLLATING_SEQUENCE qualifier with an IGNORE option specified results in the following fields being compared as equal before tie breaking:

   ```
   252-3412
   252 3412
   2523412
   ```

2. `/COLLATING_SEQUENCE=(SEQUENCE=("A"-"L","LL","M"-"R","RR","S"-"Z"))`

   This /COLLATING_SEQUENCE qualifier defines a sequence in which the double character LL collates as a single character between L and M, and the double character RR collates as a single character between R and S. These double characters would otherwise appear in their usual alphabetical order. By default, this user-defined sequence does not define any other characters, such as lowercase a to z.

11.4 Running Sort as a Batch Job

11.4.1 Definition: Batch Jobs

Batch jobs are programs or DCL command procedures that run independently of your current session. If you are sorting large files, consider submitting the Sort operation as a batch job because the sort will require some time. See Chapter 18, Chapter 15 and Chapter 16 for more information about batch jobs and command procedures.

11.4.2 Command Procedures

Specify the SORT command in your command procedure just as you would write it on the screen. If your default directory does not contain the files to be sorted, explicitly set your default directory in the command procedure or include the directory in the command file specifications.

Example

The following example submits the DCL command procedure SORTJOB.COM as a batch job. The text of the command procedure is shown following the command line:

```
$ SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]   ! Set default to location of input files
$ SORT/KEY=(POSITION:10,SIZE:15) EMPLOYEE.LST BYNAME.LST
$ TYPE BYNAME.LST
$ EXIT
```

11.4.3 Including Input Records

You can include the input records in the batch job by placing them after the SORT command with one record per line. Individual sort records can be longer than one line.

As with terminal input of records, specify the input file parameter as SYS$INPUT. Use the /FORMAT qualifier to specify the record size in bytes and the approximate file size in blocks. Approximately six 80-character lines equal one block.

Example

The following example demonstrates including input records in a command procedure:

```
$ SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]
$ SORT/KEY=(POSITION:10,SIZE:15) -
SYS$INPUT-
/FORMAT=(RECORD_SIZE:24,FILE_SIZE:10) -
BYNAME.LST
$ DECK
BST 7828 MCMAHON JANE
ADM 7933 ROSENBERG HARRY
COM 8102 KNIGHT MARTHA
ANS 8042 BENTLEY PETER
BIO 7951 LOWELL FRANK
$ EOD
```

11.5 Merging Files

11.5.1 MERGE Command

The MERGE command combines up to 10† sorted files into one ordered output file. You can merge input files that have the same format and have been sorted by the same key fields.

By default, Merge checks the sequence of the records in the input files to be sure they are in order. Specify the /CHECK_SEQUENCE qualifier if you want Merge to check the order. If you specify this qualifier and a record is out of order (for example, if you have not sorted one of the input files), Merge reports the following error:

```
%SORT-W-BAD_ORDER, merge input is out of order
```

You can use the same qualifiers with the MERGE command as you use with the SORT command with two exceptions:

- You cannot specify a process (/PROCESS) for a Merge operation.
- The /CHECK_SEQUENCE qualifier is used only for a merge operation.

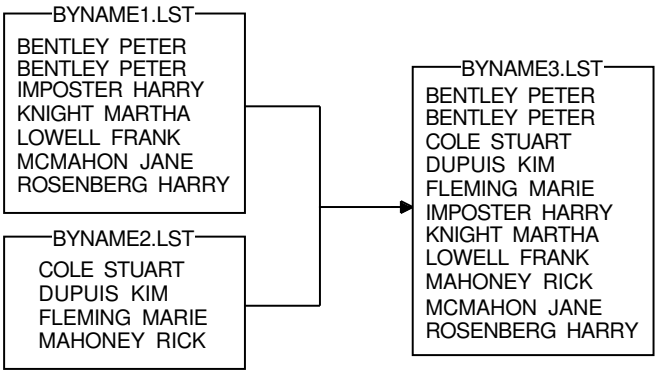
† The high-performance Sort/Merge utility supports up to 12 input files.

Example

In the following example, the files BYNAME1.LST and BYNAME2.LST have already been sorted by customer name in ascending order. The command shown merges them:

```
$ MERGE BYNAME1.LST,BYNAME2.LST BYNAME3.LST
```

The output file BYNAME3.LST contains all the records from both files, BYNAME1.LST and BYNAME2.LST, as shown in the following figure:



11.5.2 Sorted Files

To merge files that are sorted using a specific key, you must specify the same key with the /KEY qualifier on the MERGE command line.

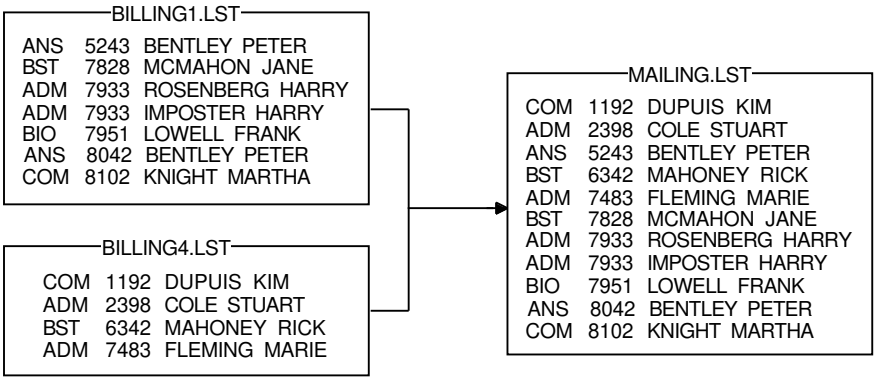
If you do not specify a key, Merge uses the default key described in Section 11.2.2.

Example

In the following example, the files BILLING1.LST and BILLING4.LST were sorted by account number (/KEY=POSITION:5,SIZE:4,DECIMAL). To merge the files into the output file MAILING.LST, enter the following command line:

```
$ MERGE/KEY=(POSITION:5,SIZE:4,DECIMAL) -
_$ BILLING1.LST,BILLING4.LST MAILING.LST
```

The results of the merge are as follows:



ZK–5063A–GE

If you want to merge files that you know are in sorted order, you can prevent sequence checking by specifying the /NOCHECK_ SEQUENCE qualifier.

11.5.3 Identical Key Fields

As with a Sort operation, when input files contain records with identical key fields, Merge does not necessarily maintain the same order in which the records had appeared in the input file. To maintain the input order of records with identical keys, specify the /STABLE qualifier on the MERGE command line. To retain only one copy of records with identical keys, specify the /NODUPLICATES qualifier.

11.6 Entering Records from a Terminal

11.6.1 Procedure

Records that you want to sort or merge do not have to be in a file. You can enter the records directly from the terminal as you enter the SORT or MERGE command. The following table describes the procedure:

Step	Task
1	Specify SYS$INPUT as the input file on the SORT or MERGE command line. Use the input file qualifier /FORMAT to specify the size of the longest record, in bytes, and the approximate size of the input file, in blocks.
2	Enter the input records on successive lines. End each record by pressing Return.
3	Press Ctrl/Z to end the file.

Example

The following example demonstrates a Sort operation in which the input records to be sorted are entered directly from the terminal:

```
$ SORT/KEY=(POSITION:8,SIZE:15) -
_$ SYS$INPUT/FORMAT=(RECORD_SIZE:24,FILE_SIZE:10) BYNAME.LST
BST 7828 MCMAHON JANE Return
ADM 7933 ROSENBERG HARRY Return
COM 8102 KNIGHT MARTHA Return
ANS 8042 BENTLEY PETER Return
BIO 7951 LOWELL FRANK Return
Ctrl/Z
```

This sequence of commands creates the output file BYNAME.LST, which contains the sorted records.

11.7 Using a Sort/Merge Specification File

11.7.1 Overview

Sort/Merge allows you to maintain sort definitions and to specify more complex sort criteria in **specification files**.† You can use any standard editor, or the DCL CREATE command to create a specification file.

A Sort/Merge specification file allows you to:

- Select records to be included in the Sort/Merge operation
- Reformat the records in the output file
- Use conditional keys or data
- Specify multiple record formats
- Create or modify a collating sequence
- Reassign work files
- Store frequently used Sort/Merge operations

After you complete the specification file, specify the file name using the /SPECIFICATION qualifier. The default file type for a specification file is .SRT.

11.7.2 Format

Each command in the specification file should start with a slash (/) and continuation characters are not required if a command spans more than one line.

Note

Many of the qualifiers used in the specification file are similar to the DCL qualifiers used in the Sort/Merge command line. Note, however, that the syntax of these qualifiers can be different. For example, the /KEY qualifier at DCL level has different syntax than the /KEY qualifier

† The high-performance Sort/Merge utility does not support specification files. Implementation of this feature is deferred to a future OpenVMS Alpha release.

in the specification file. See Section 11.9.4 for a summary of the specification file qualifiers.

11.7.3 Overriding a Qualifier

Any DCL command qualifiers that you specify on the command line override corresponding entries in the specification file. For example, if you specify the /KEY qualifier in the DCL command line, Sort/Merge ignores the /KEY clause in the specification file.

11.7.4 Order of Qualifiers

Generally, there is no required order in which you must specify the qualifiers in a specification file. However, the order becomes significant in the following cases:

- Sorting by more than one key field if you do not specify the NUMBER:*n* key element
- Describing the output format
- Defining multiple record types

When you specify the FOLD, MODIFICATION, and IGNORE keywords with the /COLLATING_SEQUENCE qualifier, you should specify all MODIFICATION and IGNORE clauses before any FOLD clauses. See Section 11.9.4 for more information about the /COLLATING_SEQUENCE qualifier.

11.7.5 Including Comments

You can include comments in your specification file by beginning each comment line with an exclamation point (!). Unlike DCL command lines, specification files do not need hyphens (-) to continue the line.

11.7.6 Examples: Specification File

1. This is an example of a specification file that can be used to sort negative and positive data in ascending order:

```
! Specification file for sorting negative and positive data
! in ascending order
!
/FIELD=(NAME=SIGN,POS:1,SIZ:1)  ❶
/FIELD=(NAME=AMT,POS:2,SIZ:4)   ❷
/CONDITION=(NAME=CHECK1,        ❸
          TEST=(SIGN EQ "-"))
/CONDITION=(NAME=CHECK2,        ❹
          TEST=(SIGN EQ " "))
/INCLUDE=(CONDITION=CHECK1,     ❺
          KEY=(AMT,DESCENDING),
          DATA=SIGN,
          DATA=AMT)
/INCLUDE=(CONDITION=CHECK2,     ❻
          KEY=(AMT,ASCENDING),
          DATA=SIGN,
          DATA=AMT)
```

As you examine the specification file, note the following:

❶ This command line defines a field that begins in byte 1 of the record and is 1 byte long. It assigns the field the name SIGN.

❷ This command line defines a field that begins in byte 2 of the record and is 4 bytes long. It assigns the field the name AMT.

❸ This is a condition statement. If there is a negative sign (–) in the SIGN byte, the CHECK1 condition is met.

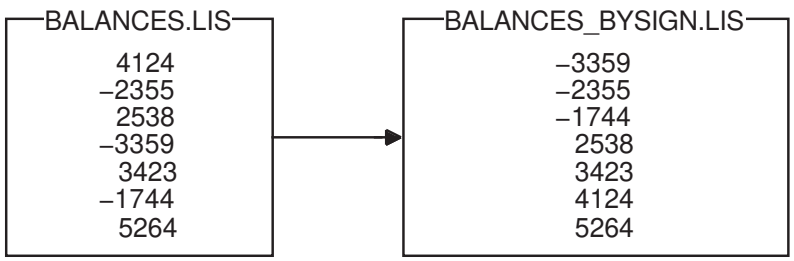
❹ This is a condition statement. If the SIGN byte is blank, the CHECK2 condition is met.

❺ If the condition CHECK1 is met, then the record is sorted in descending order.

❻ If the condition CHECK2 is met, then the record is sorted in ascending order.

Figure 11–8 shows the result of using the specification file on an input file named BALANCES.LIS.

Figure 11–8 Output from Using a Specification File



ZK–5448A–GE

2.

```
/FIELD=(NAME=RECORD_TYPE,POS:1,SIZ:1)   ! Record type, 1-byte
/FIELD=(NAME=PRICE,POS:2,SIZ:8)         ! Price, both files
/FIELD=(NAME=TAXES,POS:10,SIZ:5)        ! Taxes, both files
/FIELD=(NAME=STYLE_A,POS:15,SIZ:10)     ! Style, format A file
/FIELD=(NAME=STYLE_B,POS:20,SIZ:10)     ! Style, format B file
/FIELD=(NAME=ZIP_A,POS:25,SIZ:5)        ! Zip code, format A file
/FIELD=(NAME=ZIP_B,POS:15,SIZ:5)        ! Zip code, format B file
/CONDITION=(NAME=FORMAT_A,              ! Condition test, format A
            TEST=(RECORD_TYPE EQ "A"))
/CONDITION=(NAME=FORMAT_B,              ! Condition test, format B
            TEST=(RECORD_TYPE EQ "B"))
/INCLUDE=(CONDITION=FORMAT_A,           ! Output format, type A
            KEY=ZIP_A,
            DATA=PRICE,
            DATA=TAXES,
            DATA=STYLE_A,
            DATA=ZIP_A)
/INCLUDE=(CONDITION=FORMAT_B,           ! Output format, type B
            KEY=ZIP_B,
            DATA=PRICE,
            DATA=TAXES,
            DATA=STYLE_B,
            DATA=ZIP_B)
```

In this example, two input files from two different branches of a real estate agency are sorted according to the instructions specified in a specification file. The records in the first file that begin with an A in the first position have this format:

```
|A|PRICE|TAXES|STYLE|ZIP|
 1 2     10    15    25
```

The records in the second file that begin with a B in the first position and have the style and zip code fields reversed, as follows:

```
|B|PRICE|TAXES|ZIP|STYLE|
 1 2     10    15  20
```

To sort these two files on the zip code field in the format of record A, first define the fields in both records with the /FIELD qualifiers. Then, specify a test to distinguish between the two types of records with the /CONDITION qualifiers. Finally, the /INCLUDE qualifiers change the record format of type B to record format of type A on output.

Note that, if you specify either key or data fields in an /INCLUDE qualifier, you must explicitly specify all the key and data fields for the Sort operation in the /INCLUDE qualifier.

Also note that records that are not type A or type B are omitted from the sort.

3.

```
/COLLATING_SEQUENCE=(SEQUENCE=
("AN","EB","AR","PR","AY","UN","UL",
"UG","EP","CT","OV","EC","0"-"9"),
MODIFICATION=("'"="19"),
FOLD)
```

This /COLLATING_SEQUENCE qualifier specifies a user-defined sequence that gives each month a unique value in chronological order. For example, if you want to order a file called SEMINAR.DAT according to the date, the file SEMINAR.DAT would be set up as follows:

```
16 NOV 1983   Communication Skills
05 APR 1984   Coping with Alcoholism
11 Jan '84    How to Be Assertive
12 OCT 1983   Improving Productivity
15 MAR 1984   Living with Your Teenager
08 FEB 1984   Single Parenting
07 Dec '83    Stress --- Causes and Cures
14 SEP 1983   Time Management
```

The primary key is the year field; the secondary key is the month field. Because the month field is not numeric and you want the months ordered chronologically, you must define your own collating sequence. You can do this by sorting on the second two letters of each month–in their chronological sequence–giving each month a unique key value.

The MODIFICATION option specifies that the apostrophe (') be equated to 19, thereby allowing a comparison of '83 and 1984. The FOLD option specifies that uppercase and lowercase letters are treated as equal.

The output from this Sort operation appears as follows:

```
14 SEP 1983   Time Management
12 OCT 1983   Improving Productivity
16 NOV 1983   Communication Skills
07 Dec '83    Stress --- Causes and Cures
11 Jan '84    How to Be Assertive
08 FEB 1984   Single Parenting
15 MAR 1984   Living with Your Teenager
05 APR 1984   Coping with Alcoholism
```

See Section 11.3 for other examples of creating user-defined collating sequences.

4.

```
/FIELD=(NAME=AGENT,POSITION:20,SIZE:15)
/CONDITION=(NAME=AGENCY,
            TEST=(AGENT EQ "Real-T Trust"
            OR
            AGENT EQ "Realty Trust"))
/DATA=(IF AGENCY THEN "Realty Trust" ELSE AGENT)
```

In this example, two real estate files are being sorted. One file refers to an agency as Real-T Trust; the other refers to the same agency as Realty Trust. The /CONDITION and /DATA

qualifiers instruct Sort to list the AGENT field in the sorted output file as Realty Trust.

5.

```
/FIELD=(NAME=ZIP,POSITION:60,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/KEY=(IF LOCATION THEN 1
      ELSE 2)
```

In this example, all the records with a zip code of 01863 will appear at the beginning of the sorted output file. The conditional test is on the ZIP field, defined with the /FIELD qualifier; the condition is named LOCATION. The values 1 and 2 in this /KEY qualifier signify a relative order for those records that satisfy the condition and those that do not.

6.

```
/FIELD=(NAME=ZIP,POSITION:60,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/DATA=(IF LOCATION THEN "NORTH CHELMSFORD"
       ELSE "Outside district")
```

In this example, the /CONDITION qualifier tests for the 01863 zip code. The /DATA qualifier specifies that the name of town field will be added to the output record, depending on the test results.

7.

```
/FIELD=(NAME=FFLOAT,POS:1,SIZ:0,F_FLOATING)
/CONDITION=(NAME=CFFLOAT,TEST=(FFLOAT GE 100))
/OMIT=(CONDITION=CFFLOAT)
```

In this example, the number 100 is considered to be an F_ FLOATING data type because field FFLOAT is defined as F_FLOATING in the /FIELD qualifier.

8.

```
/FIELD=(NAME=AGENT,POSITION:1,SIZE:5)
/FIELD=(NAME=ZIP,POSITION:6,SIZE:3)
/FIELD=(NAME=STYLE,POSITION:10,SIZE:5)
/FIELD=(NAME=CONDITION,POSITION:16,SIZE:9)
/FIELD=(NAME=PRICE,POSITION:26,SIZE:5)
/FIELD=(NAME=TAXES,POSITION:32,SIZE:5)
/DATA=PRICE
/DATA="  "
/DATA=TAXES
/DATA="  "
/DATA=STYLE
/DATA="  "
/DATA=ZIP
/DATA="  "
/DATA=AGENT
```

The /FIELD qualifiers define the fields in the records from an input file that has the following format:

```
AGENT ZIP STYLE CONDITION PRICE TAXES
```

The /DATA qualifiers, which use the field-names defined in the /FIELD qualifiers, reformat the records to create output records of the following format:

```
PRICE TAXES STYLE ZIP AGENT
```

11.8 Optimizing a Sort or Merge Operation

11.8.1 Overview

There are several ways in which you can improve the efficiency of a Sort or Merge operation, based on your sorting environment. Use the /STATISTICS qualifier with the SORT or MERGE command to get information about the variables in your sorting environment.†

After you examine the statistics display, consider any of the optimization options presented in the following sections.

Example

When you enter the SORT or MERGE command with the /STATISTICS qualifier, you see output similar to the following:

```
$ SORT/STATISTICS PAGEANT.LIS DOCUMENT.LIS

                OpenVMS Sort/Merge Statistics

Records read:            3 ❶     Input record length:       26
Records sorted:          3       Internal length:           28
Records output:          3       Output record length:      26
Working set extent: 16384 ❷      Sort tree size:            42
Virtual memory:        392       Number of initial runs:     0
Direct I/O:             10       Maximum merge order:        0
Buffered I/O:           11       Number of merge passes:     0
Page faults:           158 ❸     Work file allocation:       0 ❹
Elapsed time: 00:00:00.54        Elapsed CPU:      00:00:00.03 ❺
```

As you examine the fields, note the following:

❶ Records read

Lists the number of records that were read during a Sort operation. See Section 11.8.3 for information on selectively omitting records from a Sort operation.

❷ Working set extent

Shows how many blocks are reserved to perform the sort operation. See Section 11.8.5 for information on making your working set larger.

❸ Page faults

† The high-performance Sort/Merge utility does not support this qualifier. Implementation of this feature is deferred to a future OpenVMS Alpha release.

Shows how many times the operating system has transferred parts of your process from physical memory to your paging device. See Section 11.8.5 for more information on preventing paging.

❹ Work file allocation

Shows how much disk space is reserved for your work file. See Section 11.8.4 for more information on work files.

❺ Elapsed CPU

Shows how much CPU time the operating system took to process the sort operation. See Section 11.8.2 for information on saving time by choosing different methods of sorting.

11.8.2 Sorting Process

Sort defines four processes for sorting data internally: record, tag, address and indexed.† RECORD is the default process. The type of process you choose affects the performance of the Sort operation as well as storage requirements. See the Section 11.2.8 for information about the different sort processes.

Before you select a sorting process, consider the following:

- How you will use the output file
  - Because record and tag sorting generate files that contain entire sorted records, these reordered files are ready to be used.
  - Both address- and index-sorted output files can be processed by a program written in a programming language such as Pascal, Fortran, MACRO, or C.
  - Address sorting creates a output file of pointers to the records in the input file. This list consists of binary RFAs plus a file number when sorting multiple input files. A program accesses the records by using the pointers.
  - Index sorting creates an output file containing both RFAs and key fields plus a file number when sorting multiple files. The format of these key fields is the same as in the input files. If the program needs the key field contents for a decision during future processing, select index sorting rather than address sorting.

  If you need to reorder records from one file in several ways for different purposes, store several output files from address or index sorting. Use the output files to access the records in the main file in the sorted order that you want.

† The high-performance Sort/Merge utility supports only the record process. Implementation of tag, address, and index processes is deferred to a future OpenVMS Alpha release.

- The temporary storage space available for sorting

  Tag sorting uses less temporary storage space than record sorting. Because record sorting keeps the record intact during the sort, it uses much more work space when the files are large. Address and index sorting use little temporary storage space.

- The type of input and output device used

  Record sorting is the only process that can accept input from cards, magnetic tape, and disk. Output from tag and record sorting can go to any output device. Output from address and index sorting must go to a device that accepts binary data.

- The differences in speed

  If you plan to retrieve the sorted records at some point in the operation, record sorting is usually the fastest process. Otherwise, address and index sorting are the fastest processes.

11.8.3 Omitting Records and Fields

From a specification file, you can improve Sort efficiency by using the /CONDITION, /INCLUDE, and /OMIT qualifiers to process only those records needed in the output file.† You can also use specification file qualifiers to reformat records, omitting unnecessary fields from the output file. These qualifiers are not available as command line qualifiers.

11.8.4 Work Files

During a Sort operation, records from the input file are read into memory. If the allocated memory cannot hold all the records, Sort transfers the sorted data to one or more temporary work files. Merge does not use work files.

You can increase sort efficiency by changing the number of work files and by assigning them to specific devices:

- The Sort command line qualifier /WORK_FILES=*n* overrides the number of work files allocated.
- Normally, Sort places work files on the device SYS$SCRATCH and accesses them in an arbitrary order. You can assign work files to specific devices in two ways:
  - In a specification file, the /WORK_FILES=(*device*,...) qualifier places the work files on the specified devices. See Section 11.9.4 for more information about using the /WORK_FILES qualifier in a specification file.
  - If you are not using a specification file, you can use the DCL command ASSIGN to assign the work files to specific devices.

† The high-performance Sort/Merge utility does not support specification files. Implementation of this feature is deferred to a future OpenVMS Alpha release.

Sort uses the SORTWORK*n* logical names to identify user-specified device names for the workfiles, where *n* is a value from 0 through 9.‡ Define a SORTWORK*n* logical as follows:

ASSIGN device: SORTWORKn

For example,

```
$ ASSIGN WORK$2: SORTWORK1
$ ASSIGN WORK$3: SORTWORK2
```

This example defines SORTWORK1 as the device WORK$2: and SORTWORK2 as the device WORK$3:. For more information on logical names, see Chapter 13.)

Consider the following when you assign work files to devices:

- Assign work files to the fastest devices available. For example, random-access, mass storage devices such as disks.
- Choose devices with the least activity and the most space available.
- Assign each work file to a different physical devices to maximize overlapping input and output.

11.8.5 Working Set Extent

If Sort requires work files (for example, if you are sorting a large file), a larger working set can increase sort efficiency. However, if your system is used heavily, it might be unable to allocate all the pages in the working set extent to your process. This can result in paging, which occurs when the operating system transfers parts of a process between physical memory and memory on a paging device; only the active part of the process remains in the physical memory. To avoid excessive paging, you can decrease the working set extent for your process. (Use the SET WORKING_ SET command to decrease the working set extent.)

11.9 Summary of Sort/Merge Qualifiers

11.9.1 Command Qualifiers

This list describes command qualifiers used with the SORT and MERGE commands. To use a command qualifier, include the qualifier immediately after the SORT or MERGE command.

/[NO]CHECK_SEQUENCE

(Applies to the MERGE command only.) Verifies the sequence of the records in MERGE input files. By default, Merge checks the sequence of records.

You can also use the /CHECK_SEQUENCE qualifier to check whether the records of one or more files (up to 10†) have been sorted. (The records will still be directed to an output file, which you must specify.) If you are checking whether

‡ For the high-performance Sort/Merge utility, *n* is a value from 0 to 254.
† The high-performance Sort/Merge utility supports up to 12.

records are sorted on a key field other than the entire record, you must specify key information, along with the requesting sequence.

Example:

```
$ MERGE/KEY=(SIZE:4,POSITION:3)/NOCHECK_SEQUENCE -
_$ PRICE1.DAT,PRICE2.DAT PRICE.LIS
```

The /NOCHECK_SEQUENCE qualifier specifies that the sequence of the input files, PRICE1.DAT and PRICE2.DAT, does need not be checked.

/COLLATING_SEQUENCE=sequence

Selects one of three predefined collating orders for character key fields, or specifies the name of a National character set (NCS) collating sequence to be used in comparing character keys.‡ Sort can arrange characters in ASCII (default), EBCDIC, or Multinational sequences.

Example:

```
$ SORT/COLLATING_SEQUENCE=MULTINATIONAL -
_$ NAMES.DAT,NOM.DAT LIST.LIS
```

This SORT command arranges the input files NAMES.DAT and NOM.DAT according to the Multinational collating sequence to create the output file LIST.LIS.

/[NO]DUPLICATES

By default, Sort retains all multiple records with duplicate keys. The /NODUPLICATES qualifier eliminates all but one of multiple records with duplicate keys. The retained records may not appear in the same order as they appeared in the input file. If you want to specify which duplicate record to keep, invoke Sort at the program level and specify an equal-key routine.

The /STABLE and the /NODUPLICATES qualifiers are mutually exclusive.

Example:

```
$ SORT/KEY=(POSITION:3,SIZE:5,DECIMAL)/NODUPLICATES -
_$ ACCT1,ACCT2 ACCT.LIS
```

This SORT command arranges the two input files according to the key supplied and eliminates all but one of multiple records with equal keys.

/KEY=(POSITION:n,SIZE:n[,field,...])

Describes key fields, including the position, size, sorting order (ASCENDING or DESCENDING), priority (NUMBER:n), and data type (such as character, binary, h_floating). By default,

‡ The high-performance Sort/Merge utility does not support the National Character Set (NCS) collating sequences. Support for NCS collating sequences is deferred to a future OpenVMS Alpha release.

Sort reorders a file by sorting entire records with character data in ascending order.

See Section 11.2.3 for detailed information about the /KEY qualifier.

/PROCESS=type

(Applies to the SORT command only.) Defines the internal sorting process. The /PROCESS qualifier allows you to choose one of four processes: record, tag, address, or index.†

See Section 11.2.8 for detailed information about the /PROCESS qualifier.

Example:

```
$ SORT/KEY=(POS:40,SIZ:2,DESC)/PROCESS=TAG YRENDAVG.DAT -
_$ DESCYRAVG.LIS
```

This Sort operation uses a tag sorting process to create the output file DESCYRAVG.LIS.

/SPECIFICATION=filespec†

Identifies a Sort or Merge specification file to be used in a Sort or Merge operation. The default specification file type is .SRT.

See Section 11.7.1 and Section 11.9.4 for information about using specification files.

/[NO]STABLE

By default, records with equal keys are not guaranteed to be placed in the output file in the order they appear in the input file. The /STABLE qualifier maintains the records in that order.

The /STABLE and /NODUPLICATES qualifiers are mutually exclusive.

Example:

```
$ SORT/KEY=(POS:1,SIZ:5,DECIMAL)/STABLE PRICESA.DAT, -
_$ PRICESB.DAT,PRICESC.DAT SUMMARY.LIS
```

In this Sort operation, records with equal keys from PRICESA.DAT will be listed first, followed by those from PRICESB.DAT, followed by those from PRICESC.DAT.

/[NO]STATISTICS†

Displays a statistical summary to SYS$OUTPUT that can be used for optimization. To save these statistics in a file, use the following command:

```
$ DEFINE/USER SYS$ERROR output-file
```

† The high-performance Sort/Merge utility supports only the record process. Implementation of tag, address, and index processes is deferred to a future OpenVMS Alpha release.

† The high-performance Sort/Merge utility does not support this qualifier. Implementation of this feature is deferred to a future OpenVMS Alpha release.

The statistical summary contains the following information:

Statistic	Description
Records read	The number of records read by Sort or Merge.
Records sorted	The number of records that have been processed using Sort. This number could be less than the number of records read if a specification file is used to select only certain records for the Sort or Merge operation.
Records output	The number of records written to the output file. This number could be less than the number of records sorted if /NODUPLICATES was selected or if I/O errors occurred when the output records were being written.
Working set extent	The number of pages in the process working set extent. This value is used as an upper limit on the size of the sort data structure. Adjusting this value is one way to improve the efficiency of a Sort operation.
Virtual memory	The number of pages of virtual memory added to the Sort image to hold the data.
Direct I/O + buffered I/O	This total is the number of I/O movements needed to read and write data. The lower this total value is, the more efficient the ordering operation.
Page faults	Indicates how well the data fits into memory: the higher the number of page faults, the less efficient the ordering operation.
Elapsed time	The total wall clock time used by the Sort or Merge operation in hours, minutes, seconds, and hundredths of seconds.
Input record length	This value is obtained from the Record Management Services (OpenVMS RMS) unless the user supplies it.
Internal length	The size in bytes of an internal format node. This includes any keys, data, a word to store the length, record file addresses (RFAs), and converted keys.
Output record length	The length of the output record. The length is computed from the input record length, the sort process, and the record reformatting requested.
Sort tree size	The number of records that fit in the Sort internal data structure.

Statistic	Description
Number of initial runs	One indication of how well the data fits into memory.
Maximum merge order	The maximum number of sorted strings that are merged at one time.
Number of merge passes	The number of times the Sort utility merges strings until one sorted output string is produced. The number of initial runs and the number of merge passes indicate how well the data fits into memory. The higher these numbers, the further the working set size is from containing the data and the longer the sorting takes.
Work file allocation	The number of blocks used for the work files. When more than one merge pass is needed, this size is approximately twice the size of the input file allocation.
Elapsed CPU	The CPU time used by the ordering operation; it does not include time spent waiting for I/O operations to complete or time spent waiting while another process executes.

Example:

```
$ SORT/STATISTICS PRICE1.DAT,PRICE2.DAT PRICE.LIS
```

This SORT /STATISTICS command results in the following statistical display:

```
                OpenVMS Sort/Merge Statistics

Records read:          793   Input record length:     80
Records sorted:        793   Internal length:         80
Records output:        793   Output record length:    80
Working set extent:    100   Sort tree size:         412
Virtual memory:        433   Number of initial runs:   2
Direct I/O:             22   Maximum merge order:      2
Buffered I/O:            9   Number of merge passes:   1
Page faults:          3418   Work file allocation:   114
Elapsed time: 00:00:05.98   Elapsed CPU:    00:00:03.63
```

/WORK_FILES[=n]

(Applies to the SORT command only.) Increases the number of Sort work files by any number, from 1 to 10† inclusively, to make each work file smaller. If the available disks are too small or too full for work files, increasing the number of files can improve the efficiency of the Sort operation.

Sort does not create work files until it needs them. If Sort needs work files, it creates two by default (SORTWORK0, SORTWORK1), which are placed in the SYS$SCRATCH directory.

† The high-performance Sort/Merge utility supports up to 255.

Example:

```
$ ASSIGN DRA5: SORTWORK0
$ ASSIGN DB0: SORTWORK1
$ ASSIGN DB1: SORTWORK2
$ SORT/KEY=(POS:1,SIZ:80)/WORK_FILES=3 -
_$ STATS1,STATS2,STATS3,STATS4 SUMMARY.LIS
```

Because the input files in this Sort operation are large files, specifying three work files improves the efficiency of the sort operation.

Note that you can also assign the work files to a specific directory on a device by including the directory name. For example, to assign SORTWORK0 to the [WORKSPACE] directory on DRA5, enter the following command:

```
$ ASSIGN DRA5:[WORKSPACE] SORTWORK0
```

11.9.2 Input File Qualifier

The following input qualifier should be included immediately after the input file specification in the SORT or MERGE command line:

/FORMAT=(RECORD_SIZE:n,FILE_SIZE:n)

Defines input file characteristics; allows you to specify or override record or file size. It must be specified immediately after the input file specification in the Sort or Merge command line.

Sort uses input file size information to determine the amount of memory needed, as well as the size of the work files for the Sort operation. If the file size is unknown (for example, you are sorting files that do not reside on disk or standard ANSI magnetic tape), Sort assumes a fairly large file size.

Specify the following qualifier values:

RECORD_SIZE:*n* — Specifies the input file's longest record length (LRL) in bytes. The maximum longest record length that can be specified depends on the file organization:

Sequential	32,767
Relative	16,383
Indexed-sequential	16,362

These values include control bytes for variable records with fixed-length control (VFC) format.

FILE_SIZE:*n* — Specifies input file size in blocks. The maximum file size accepted is 4,294,967,295 blocks.

You can also use /FORMAT as an output file qualifier. See Section 11.9.3 for more information.

Example:

```
$ SORT/KEY=(POS:40,SIZ:2,DESC) -
_$CRA0:YRENDAVG.DAT/FORMAT=(RECORD_SIZE:41,FILE_SIZE:3) -
_$DESCYRAVG.LIS
```

Because the input file YRENDAVG.DAT does not reside on a disk device or ANSI magnetic tape, file organization must be described by the /FORMAT qualifier.

11.9.3 Output File Qualifiers

The following output qualifiers can be used with the SORT and MERGE commands. To use an output file qualifier, include the qualifier immediately after the output file specification in the SORT or MERGE command line.

/ALLOCATION=n

Specifies the number of blocks, from 1 through 4,294,967,295, to be preallocated to the output file for optimization. Use this qualifier when you know that the output file allocation will differ substantially from the total input file allocation (for example, when reformatting data or omitting records).

The /ALLOCATION qualifier is required if the /CONTIGUOUS qualifier is used.

Example:

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT -
_$ SUMMARY.LIS/ALLOCATION=1000/CONTIGUOUS
```

This SORT command allocates 1000 contiguous blocks for the output file SUMMARY.LIS.

/BUCKET_SIZE=n

Specifies OpenVMS RMS bucket size (the number of 512-byte blocks per bucket) to be used by relative and indexed sequential output disk files for optimization. A value of 1 through 32 is allowed.

If the output file organization is the same as for the input files, the default value is the same as the bucket size of the first input file. If output file organization is different, the default value is 1.

Example:

```
$ SORT/KEY=(POS:1,SIZ:80) STATS1.DAT,STATS2.DAT -
_$ SUMMARY.LIS/BUCKET_SIZE=16/RELATIVE
```

This SORT command results in the output file SUMMARY.LIS that has a bucket size of 16 with relative organization.

/CONTIGUOUS

Requests that the output file be stored in contiguous disk blocks to decrease access time. Must be used with the /ALLOCATION qualifier. By default, Sort/Merge does not allocate contiguous disk blocks for the output file.

Example:

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT -
_$ SUMMARY.LIS/ALLOCATION=1000/CONTIGUOUS
```

This SORT command allocates 1,000 contiguous blocks for the output file SUMMARY.LIS.

/FORMAT=(type:n[,...])

Specifies the output file record format (FIXED:n, VARIABLE:n, or CONTROLLED:n) if it differs from the input file format. You can also specify the size (SIZE:n) or the block size (BLOCK_SIZE:n) of the file records.

If the Sort operation is a record or tag sort, the default output record format is the same as the first input file record format. If the Sort operation is an address or index sort, the default output record format is fixed record format. If the input files have different record formats, Sort provides an output record size that is large enough to contain the largest record in the input files.

You can specify the following qualifier values.

BLOCK_SIZE:*n*	Specifies the output file's block size, in bytes, if you have directed the file to magnetic tape. If the input file is a tape file, the block size of the output file defaults to that of the input file. Otherwise, the output file block size defaults to the size used when the tape was mounted. Acceptable values for *n* range from 20 to 65,532. To ensure correct data interchange with other Digital systems, however, specify a block size of not more than 512 bytes. For compatibility with systems that are not made by Digital, the block size should not exceed 2,048 bytes.
CONTROLLED:*n*	Specifies variable with fixed-length control (VFC) records in the output file.
FIXED:*n*	Specifies fixed-length records in the output file.
SIZE:*n*	Specifies the size, in bytes, of the fixed portion of VFC (CONTROLLED) records, up to a maximum of 255 bytes. If you do not specify SIZE, the default is the size of the fixed portion of the first input file. If you specify this size as 0, OpenVMS RMS defaults the value to 2 bytes.
VARIABLE:*n*	Specifies variable-length records in the output file.

For any qualifier value, you can optionally specify *n* as the maximum record size (in bytes) of the output records. The maximum record size allowed depends on the file organization:

Sequential files	32,767
Relative files	16,383
Indexed-sequential files	16,362

These maximum record size values include control bytes for variable records with fixed-length control (VFC) format.

Example:

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/FORMAT=FIXED:80
```

The input file STATS.DAT consists of variable-length records that are 80 bytes in length. The /FORMAT qualifier specifies that the output file, SUMMARY.LIS, consists of fixed-length records.

/INDEXED_SEQUENTIAL†

Defines the file organization for the output file as indexed sequential. Note that the output file must already exist and must be empty. In addition, you must specify that the empty file is to be overlaid with the sorted records by using the /OVERLAY qualifier.

Example:

```
$ CREATE/FDL=NEW.FDL AVERAGE.DAT
$ SORT/KEY=(POS:1,SIZ:80) DATA.DAT,STATS.DAT -
_$ AVERAGE.DAT/INDEXED_SEQUENTIAL/OVERLAY
```

The CREATE/FDL command creates the empty file AVERAGE.DAT. The SORT command specifies that the output file have an indexed-sequential organization and be written to the empty file AVERAGE.DAT.

/OVERLAY‡

Specifies an existing empty file that the output file is to be overlaid on, or written to. The /OVERLAY qualifier is required when you use the /INDEXED_SEQUENTIAL qualifier.

If the input file organization is indexed-sequential, the output file must already exist and be empty. If the output file is not empty, /OVERLAY does not write over the file. Instead, it appends the result of the sort to the existing output file.

You can use the CREATE/FDL utility to create an empty data file. Any attributes that you specify when creating the empty file then become attributes of the Sort output file.

† The high-performance Sort/Merge utility does not support indexed sequential output file organization. Implementation of this feature is deferred to a future OpenVMS Alpha release.

‡ The high-performance Sort/Merge utility does not support use of the /OVERLAY qualifier. Support of this feature is deferred to a future OpenVMS Alpha release.

Example:

```
$ CREATE/FDL=NEW.FDL AVERAGE.DAT
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT AVERAGE.DAT/OVERLAY
```

The FDL file NEW.FDL specifies special attributes for the file AVERAGE.DAT. When Sort writes output to that file, the resulting Sort output file has the attributes specified by the FDL file.

/RELATIVE

Defines the file organization for the output file as relative.

Example:

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/RELATIVE
```

Because the input file STATS.DAT is not a relative file and the output file, SUMMARY.LIS, will be, /RELATIVE qualifies the output file specification.

/SEQUENTIAL

Defines the file organization for the output file as sequential. This is the default for address and index sorting operations. The default for record and tag sorting operations is the organization of the first input file.

Example:

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/SEQUENTIAL
```

Because the input file STATS.DAT is not a sequential file and the output file SUMMARY.LIS will be, /SEQUENTIAL qualifies the output file specification.

11.9.4 Specification File Qualifiers

The following qualifiers can be used in specification files.† Note that these qualifiers are valid only within a Sort/Merge specification file.

/CDD_PATH_NAME=‘‘cdd-path-name’’

Identifies fields and attributes defined for use with the Common Data Dictionary (CDD/Plus) using the CDD/Repository command. Once the fields have been identified, they can then be used later with other specification file qualifiers, such as /KEY, /CONDITION, /INCLUDE, or /OMIT.

/CDD_PATH_NAME can be used in place of or in conjunction with /FIELD statements.

The ‘‘cdd-path-name’’ value is the CDD/Plus record definition within CDD/Plus. You can use the /CDD_PATH_NAME qualifier only if your system has CDD/Plus installed.

† The high-performance Sort/Merge utility does not support specification files. Implementation of this feature is deferred to a future OpenVMS Alpha release.

Example:

```
/CDD_PATH_NAME="customer"
```

The /CDD_PATH_NAME qualifier identifies the customer record, which had been defined previously in CDD/Plus.

/[NO]CHECK_SEQUENCE

(Applies to the MERGE command only.) Specifies whether or not the sequence of records in the input file is checked. By default, Merge does not check the sequence of records.

Example:

```
/NOCHECK_SEQUENCE
```

The /NOCHECK_SEQUENCE qualifier overrides the Merge utility's default behavior.

/COLLATING_SEQUENCE=(SEQUENCE=sequence-type
[,MODIFICATION=(''char1'' operator ''char2'')]
[,IGNORE=character or character range,...]
[,FOLD]
[,[NO]TIE_BREAK])

Specifies one of three predefined collating sequences (ASCII, EBCDIC, or Multinational) or a user-defined sequence for character key fields. Allows you to modify any of the predefined collating sequences or any previously defined user-defined sequences.

See Section 11.3 for information about using the ASCII, EBCDIC, and Multinational collating sequences.

You can specify the following qualifier values:

SEQUENCE — Specification files support the ASCII, EBCDIC, multinational, and user-defined collating sequences. See Section 11.3 for information about these collating sequences.

MODIFICATION — Specifies a change to the collating sequence specified in the SEQUENCE option. You can modify the ASCII, EBCDIC, Multinational, or user-defined sequence. The sequence being modified must be specified with the SEQUENCE qualifier even if the sequence is the default (ASCII).

- character — Specifies a character in the collating sequence.
- operator — Specifies the operator used to compare the characters. You can specify greater than (>), less than (<), or equal to (=).

The following kinds of changes are permitted in the MODIFICATION option:

- A single or double character can be equated to a single character that has already been assigned a collating value ("a"="A").
- A single or double character can collate after a single character that has already been assigned a collating value ("CH">"C").
- A single or double character can collate before a single character that has already been assigned a collating value ("D"<"A").
- A double character can be equated to a previously defined double character ("CH" = "SH").
- A single character can be equated to a double character sequence ("C" = "CH").

IGNORE

Specifies that Sort/Merge ignore a character or character range in the collating sequence when making an initial comparison. Note that, when tie-breaking takes place, Sort /Merge considers the characters specified with the IGNORE value.

FOLD

Specifies that all lowercase letters be given the collating value of their uppercase equivalents. For ASCII, EBCDIC, and user-defined sequences, the lowercase letters are a to z.

Because the lowercase letters in the Multinational sequence already have the collating value of their uppercase equivalents, using FOLD is unnecessary.

[NO]TIE_ BREAK

Specifies whether or not Sort/Merge should use numeric values to break any ties between characters that have equivalent values. By default, tie-breaking occurs with the Multinational sequence. Specifying NOTIE_BREAK overrides this default and ensures that no further comparisons are made after the initial comparison.

A TIE_BREAK option must be specified for the ASCII, EBCDIC, and user-defined sequences in order for tie-breaking to occur. TIE_ BREAK should be used when specifying the FOLD or MODIFICATION value for the these sequences.

Examples:

See Section 11.3 and Section 11.7 for examples of the use of collating sequences in specification files.

/CONDITION=(NAME=condition-name, TEST=(field-name operator test-condition [logical-operator...]))

A specification file can be used to change the relative order of a record or to alter the contents of certain fields in a record. You must first use the /CONDITION qualifier to define a conditional test. Once you define a test using the /CONDITIONAL qualifier, you can use that same test with the /KEY or /DATA qualifier to change the order of record. You can also use the test with the /OMIT or /INCLUDE qualifier to change the contents of a record.

If you want to change the order of records in the output file, first specify a condition name with the /CONDITION qualifier and set up a test for what meets that condition. Then, specify the relative order with the /KEY qualifier of the form:

```
/KEY=(IF condition-name THEN value ELSE value)
```

You can use any values to specify the relative order of the records.

The /CONDITION qualifier also permits you to change the contents of a field in the output records. First specify a condition name, and then set up a test for what meets the condition. Specify the contents you want in the field in a /DATA qualifier of the form:

```
/DATA=(IF condition-name THEN "new-contents"
       ELSE "new-contents")
```

You can specify the following qualifier values:

NAME Specifies the name of the condition that you are testing. This condition-name can be used in /KEY, /DATA, /OMIT, and /INCLUDE qualifiers after it has been defined using the /CONDITION qualifier.

TEST Specifies the conditional test.

field-name
Specifies the name of the field you are testing. The field-name must be defined previously by the /FIELD qualifier.

operator
Specifies the logical or relational operator used in the conditional test. The logical operators that you can use are AND and OR. The relational operators that you can specify are as follows:

EQ Equal to

NE Not equal to

GT Greater than

GE Greater than or equal to

LT Less than

LE Less than or equal to

test-condition
Specifies the constant or field-name against which you are testing. A constant is specified with the following format:

Decimal_digits (default)
%Ddecimal_digits
%Ooctal_digits
%Xhexadecimal_digits
"character"

Normally, you do not need to specify the radix operator (%D); however, test-condition will assume the same data type as the field-name.

Examples:

See Section 11.7 for examples of the use of the /CONDITION qualifier in specification files.

/DATA=field-name
/DATA=(IF condition THEN ‘‘new contents’’ ELSE ‘‘new contents’’)

Use the /DATA qualifier to eliminate or reorder fields from the output record. Specify the data fields in the order you want them to appear in the output record. A /DATA qualifier must identify every field in the records you are directing to the output file. Only those fields identified by the /DATA qualifier to be directed to the output file.

You can conditionally change the contents of a field in the output records by first specifying a condition name and then setting up a test for what meets the condition in a /CONDITION qualifier. You then specify the contents you want in the field in a /DATA qualifier of the form:

```
/DATA=(IF condition-name THEN "new-contents" ELSE "new-contents")
```

You can specify the following qualifier values:

field-name	Specifies the name of a field in a record. The field-name must be defined previously in a /FIELD qualifier.
condition-name	Specifies a condition-name that has been defined previously in a /CONDITION qualifier.
new-contents	Specifies how the record is to be altered. The new-contents can be a constant or a field-name that has been defined in a /FIELD qualifier.

Examples:

See Section 11.7 for examples of the use of the /DATA qualifier in specification files.

/FIELD=(NAME=field-name,POSITION:n,SIZE:N,
[DIGITS:n,]data-type
/FIELD=(NAME=field-name,VALUE:n,SIZE:N,
[DIGITS:n,]data-type)

Defines the fields in the input files when you are altering the order or format of output records. These fields include key fields, fields to be compared, and fields to be directed to the output file. You identify each field by specifying a name, its position and size in the record, and its data type.

Field names must be unique; no duplicate field names are allowed. In addition, you cannot use more than 255 field definitions.

You can also use /FIELD to define a constant and assign it a value of any valid Sort/Merge data type for use in /CONDITION, /DATA, and /KEY statements.

You can specify the following qualifier values:

NAME	Specifies the name of the field. The field-name cannot have any embedded spaces, must begin with an alphabetic character, and can be no longer than 31 characters.
POSITION:*n*	Specifies the position of the field in the record.
VALUE:*n*	Assigns a value to a constant field for use in a /CONDITION, /DATA, or /KEY statement. If you specify VALUE:*n*, do not specify /POSITION:*n* because the field is a constant and not part of an input record.
SIZE:*n*	Specifies the size of a field containing character or binary data. In the specification file, SIZE implies byte lengths. The data type determines what values are acceptable, as follows: – For character data, the size must not exceed 32,767 characters. – For binary data, the size specified must be 1, 2, 4, 8, or 16 bytes. – For floating-point data, no size is specified.

DIGITS:*n* Specifies the size of a field containing decimal data. The size of a field containing decimal data must not exceed 31 digits. Note that DIGITS:n is used only when describing a field containing decimal data.

data-type Specifies the data type of the field. You are not required to specify the data-type if it is character; Sort assumes character data type by default. See Section 11.2.3 for a list of the data types recognized by Sort/Merge.

Example:

```
/FIELD=(NAME=SALARY,POSITION:10,DIGITS:8,DECIMAL)
```

This /FIELD qualifier identifies a field in a record by the name SALARY, specifies that it starts in position 10 of the record, is 8 digits long, and consists of decimal data.

/INCLUDE=(CONDITION=condition[,KEY=...] [,DATA=...])

You can specify that records are to be conditionally included in an output file. After defining a condition in a /CONDITION qualifier, specify record selection in an /INCLUDE qualifier requesting that records satisfying the condition are to be included in the output file.

You can specify multiple /INCLUDE and /OMIT qualifiers in a specification file. The order in which you specify them determines the order the input records are tested for inclusion. After the last /INCLUDE qualifier, all records that have not already been included or explicitly omitted are omitted.

You can unconditionally include any records not previously omitted or included by specifying the /INCLUDE qualifier without a condition.

When sorting multiple record formats, one /INCLUDE qualifier should be specified for each different record format among the records to be sorted. If you do not specify a KEY option within the INCLUDE qualifier, Sort assumes the default key definitions. If the KEY is specified in the /INCLUDE qualifier, the default key definitions are not used. The order of the KEY fields in the /INCLUDE qualifier determines how the internal key is built for sorting. The order of the DATA fields in the /INCLUDE qualifier determines the way the output record is formatted. If you specify a key or data field in an /INCLUDE qualifier, you must define all other key or data fields in the record.

You can specify the following qualifier values:

CONDITION Refers to the condition-name specified in a previous /CONDITION qualifier.

KEY Defines a key field because the default record type defined in the /KEY qualifier is not being used.

DATA Defines a data field because the default record type defined in the /DATA qualifier is not being used.

Example:

```
/FIELD=(NAME=ZIP,POSITION:20,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/INCLUDE=(CONDITION=LOCATION)
```

These /CONDITION and /INCLUDE qualifiers specify that records with the zip code 01863 will be included in the output file.

/KEY=field-name
/KEY=(field-name,order)
/KEY=([IF condition THEN value ELSE]...) value [,order]

Specify the key fields to be used in the Sort operation. If you are sorting the entire record using character data, there is no need to specify your key field. Otherwise, specify a /KEY qualifier for each of the keys, in the order of their priority. You can sort on as many as 255 key fields.

There are three ways to use the /KEY qualifier:

- To identify the key field name.
- To identify the key field name and to specify sorting order. In this case, enclose the field name and the order option in parentheses.
- As a conditional qualifier, to change the order of records in the output file. First, specify a condition name in a /CONDITION qualifier, and set up a test for what meets that condition. Then, specify the relative order in a /KEY qualifier of the form:

  ```
  /KEY=(IF condition-name THEN value ELSE value)
  ```

  You can use any values to specify the relative order of the records.

You can specify the following qualifier values:

field-name	Specifies the name of the key field. The field-name has been previously specified in a /FIELD qualifier.
order	Specifies the order of the sort. The ASCENDING option specifies ascending order for a Sort or Merge operation. This option is the default. The DESCENDING option specifies descending order for a Sort or Merge operation.
value	Specifies the key. The value can be a constant or a field-name that has been defined in a /FIELD qualifier.

Examples:

1.

```
/FIELD=(NAME=SALARY,POSITION:10,DIGITS:8,DECIMAL)
/KEY=(SALARY,DESCENDING)
```

This /KEY qualifier specifies that the key field is SALARY and that the sorting order is descending.

2.

```
/FIELD=(NAME=ZIP,POSITION:20,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/KEY=(IF LOCATION THEN 1
      ELSE 2)
```

In this example, all the records with the zip code 01863 are to appear at the beginning of the sorted output file. The conditional test LOCATION (defined in a /CONDITION qualifier) is on the ZIP field (named in a /FIELD clause). The values of 1 and 2 in this /KEY clause signify a relative order for those records that satisfy the condition and those that do not.

/OMIT=(CONDITION=condition-name)

Specifies that records are to be omitted from the output file based on a condition defined with a /CONDITION qualifier.

First, you must define a condition with the /CONDITION qualifier. Specify your records with an /OMIT qualifier to request any records that satisfy the condition be omitted from your Sort operation. By default, Sort/Merge includes all the other input records in the output file.

You can specify multiple /OMIT and /INCLUDE qualifiers in your specification file. The order in which you specify them determines the order in which the input records are tested for omission. All the records that have not already been included or omitted after the last /OMIT qualifier are included. You can unconditionally omit any records not previously omitted or included by specifying the /OMIT qualifier only.

Example:

```
/FIELD=(NAME=ZIP,POSITION:20,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/OMIT=(CONDITION=LOCATION)
```

These /CONDITION and /OMIT qualifiers specify that records with the zip code 01863 are to be omitted from your output file.

/PAD=single-character

Specifies the character Sort will use to expand, or ‘‘pad,’’ a string when reformatting records or when comparing strings of unequal length. By default, Sort uses the null character for padding, ensuring conformity with the previous versions. Double characters that can be defined as single characters ("ch" > "c") cannot be used as pad characters. Characters, decimal, octal, or hexadecimal digits can be used.

The pad character should be specified as follows:

- Use quotation marks for a character. For example, " # " would specify the number sign.
- Use decimal radix for decimal digits. For example, %D35 would specify the decimal number 35.
- Use octal radix for octal digits. For example, %O043 would specify the octal number 043.
- Use hexadecimal radix for hexadecimal digits. For example, %X23 would specify the hexadecimal number 23.

Example:

```
/PAD="."
```

This example of a /PAD qualifier specifies that records will be padded with periods.

/PROCESS=type

(Applies to the SORT command only.) Defines the processing method (record, tag, address, or index) for the sorting operation. If you intend to reformat the output records, you cannot use address or index sort. Specify the process type as RECORD, TAG, ADDRESS, or INDEX.

See Section 11.8.2 for a comparison of the four types of process.

Example:

```
/PROCESS=tag
```

This example of the /PROCESS qualifier specifies that Sort use a tag sorting process.

/[NO]STABLE

Specifies that records with equal keys are directed to the output file in their input file order. The default condition is /NOSTABLE.

By default, when records are sorted with identical keys, the order of those records in the output file may not be the same as they appeared in the input file. Specifying the /STABLE qualifier in a specification file arranges records with equal keys in the output file in the order of the input files as specified in the command line. If you use this qualifier when sorting multiple input files, on output, records with equal keys in the first file will precede those from the second file and so on.

Example:

```
/STABLE
```

This example of the /STABLE qualifier ensures that records with equal keys will have the same order in the input and output files.

/WORK_FILES=(device[,...])

(Applies to the SORT command only.) Reassigns work files to different disk-structured devices to improve performance. Using the /WORK_FILES qualifier in a specification file makes it unnecessary to assign logicals prior to invoking Sort at the command or program level.

Unlike the DCL qualifier /WORK_FILES=n, the specification file qualifier /WORK_ FILES=(device[,...]) specifies work file assignments, not the number of work files.

See Section 11.8.4 for more information about the use of work files.

Example:

```
/WORK_FILES=("WRKD$:")
```

This example of a /WORK_FILES qualifier assigns one of Sort's work files to the device WRKD$: because that device has the most space available.

12

Devices: Using Private Tapes and Disks

12.1 Overview

This chapter describes how to allocate, initialize, and mount a device for private use. It includes information about:

- Displaying device information
- Allocating devices
- Initializing volumes
- Mounting volumes
- Accessing files on private devices
- Dismounting volumes

12.1.1 References

For additional information refer to:

- The *OpenVMS DCL Dictionary* or online help, for information about the commands discussed in this chapter
- The *OpenVMS System Management Utilities Reference Manual*, for information about the MOUNT command
- The *VMScluster Systems for OpenVMS*, for information about the devices in VAXcluster or VMScluster environments

12.2 Devices Overview

12.2.1 Device Managment

If all the files you create and use are located on devices managed by a system manager, you might not need to learn how to access a tape or disk drive. The system manager can set up **logical names** to refer to the devices you need to use. You can then read, write, and copy files to and from the device by using the logical name. In most cases, the system manager sets up and maintains devices that are shared by a group of users.

12.2.2 Accessing Devices

If you have a tape or disk drive available for your private use (for example, if your workstation includes an integral tape or diskette drive), you should be familiar with a few commands for accessing those devices. Diskettes and diskette drives are one type of disk device. For simplicity, this chapter uses the generic term *disks* to refer to all types of disks.

12.2.3 Device Security

Although a disk or tape is for your exclusive use, anyone with physical access to it can read its contents on some system. It is good practice to keep disks and tapes in a safe place and to erase sensitive data before recycling them so the next user cannot read their contents.

12.2.4 Volumes

In this chapter, storage media are sometimes referred to as **volumes**. While the storage media itself is a piece of hardware, you can think of it in terms of its stored data; a volume is a disk or tape that has an organized collection of data.

12.2.5 Setting Up Private Volumes

To perform your work on a device that cannot be accessed by other users, you can create a private volume and mount it on a device allocated exclusively for you. The steps for setting up a private volume are as follows:

Step	Task
1	Use the DCL command ALLOCATE to assign the device (disk or tape) drive to your process.
2	Use the DCL command INITIALIZE to format the disk or tape volume and write an identifying label on the volume.
3	Use the DCL command MOUNT to make a volume and the files or data it contains accessible to your process.

12.2.6 Printing Files from Private Devices

Although you can print a file from a privately owned volume, the volume containing the file to be printed must remain mounted until after the file has completed printing. If you want to dismount the volume before the file is done printing, copy the file directly to the printer. For example:

```
$ COPY MYPHILE.DAT LPA0:
```

12.3 Displaying Device Information

12.3.1 Using the SHOW DEVICES Command

To display information about volumes that are on your system, enter the SHOW DEVICES command. To obtain additional information or information about a specific device, enter the SHOW DEVICES command in one of the following ways:

- To check the densities, volume labels, UICs, and relative volume numbers of mounted volumes, enter the SHOW DEVICES/FULL command.
- To display information for all the drives of a particular type configured in the system, specify a generic device code for magnetic tape drives (such as MT).
- To display information for a volume mounted on a specific drive, specify the physical device name (for example, MTA1:).

12.3.2 Example

In the following example, the SHOW DEVICES command displays the available devices DMA1: and MTA1:

```
$ SHOW DEVICES

Device           Device         Error    Volume         Free  Trans Mnt
 Name            Status         Count     Label        Blocks Count Cnt
DMA2:           Mounted            0     BACKUP_FILE    4442     7   1
MTA1:           Online             0
    .
    .
    .
```

12.4 Allocating Devices

12.4.1 Overview

When you allocate a device, you reserve the device for exclusive use by your process. The device remains allocated to your process until you explicitly deallocate it (with the DCL command DEALLOCATE) or until you log out.

12.4.2 Format

Before you can use files or data on a private volume, you must allocate (locally assign) a disk or tape drive to your process with the DCL command ALLOCATE. The format for the ALLOCATE command is as follows:

ALLOCATE device-name[:] [logical-name]

The elements are as follows:

device-name Specifies the drive on which the volume is loaded. The name can be a physical name, a generic name, or a logical name.

logical-name Specifies an optional logical name to be associated with the device.

12.4.3 Methods of Allocating Devices

Table 12–1 summarizes the ways in which you can allocate a device.

Table 12–1 Allocating a Device

Example	Explanation

To allocate a specific physical device:
`$ ALLOCATE DMB2:`
`%DCL-I-`
`ALLOC, _MARS$DMB2: allocated`
	Uses a physical device name (DMB2) to request the allocation of a specific RK06 or RK07 disk drive (unit 2 on controller B).

(continued on next page)

Table 12–1 (Cont.) Allocating a Device

Example	Explanation

To allocate the first available device:
`$ ALLOCATE DM: DISK`
`%DCL-I-`
`ALLOC, _MARS$DMB1: allocated`
	Uses the generic device code DM to allocate the first available RK06 or RK07 disk device. In addition, creates the logical name DISK in the process logical name table and assigns it to the name of the allocated device.

To allocate a device by logical name:
`$ ALLOCATE DRIVE1: D1`
`%DCL-I-`
`ALLOC, _MARS$DBA3: allocated`
	Allocates DRIVE1 (a logical name that translates to the physical device DBA3) and assigns a new logical name D1 to the allocated device.

To select a particular device type: `$ ALLOCATE/GENERIC RK07  MYDISK`	Uses the /GENERIC qualifier to allocate a particular type of device (an RK07 disk).

To specify a list of generic devices:
`$ ALLOCATE MF,MT,MS DRIVE`
`%DCL-I-`
`ALLOC, _MARS$MTA0: allocated`
	Specifies a list of generic device names; the first available device (MTA0) is the only one to be allocated. In addition, assigns the logical name DRIVE to MTA0.
	The ALLOCATE command allocates only one device to a process. Therefore, although each element in the list represents a unique, generic device type, only one of the specified generic devices is allocated.

12.5 Initializing Volumes

12.5.1 Using the INITIALIZE Command

Initializing a disk or tape volume allows you to write files to that disk or tape. To initialize a volume, use the DCL command INITIALIZE, which does the following:

- Deletes (erases) all existing data on the volume, if any, and creates a new file structure
- Writes a label on the volume to identify it
- Defines the owner UIC and the protection for the volume

Note

The INITIALIZE command does not prevent you from initializing another user's volume; to be sure the volume you initialize is your own, allocate the device before you initialize the volume.

If you give a volume to another user for initialization (for example, if you lack sufficient privileges to initialize a particular volume), you should provide the volume label, the owner UIC, and the **protection code** for the volume.

12.5.2 INITIALIZE Command Format

The format for the INITIALIZE command is as follows:

```
INITIALIZE device-name[:] volume-label
```

The fields are as follows:

device-name	Specifies the name of the device on which the volume is physically mounted.
volume-label	Identifies the volume. You can specify up to 12 alphanumeric characters for a disk volume or up to 6 alphanumeric characters for a magnetic tape volume.

12.5.3 Initializing Disk Volumes

By default, the INITIALIZE command builds a Files–11 structure on your new volume. The default format for disk volumes initialized for or by the OpenVMS operating system is called the Files–11 On-Disk Structure Level 2. The INITIALIZE command can also initialize disk volumes in Files–11 On-Disk Structure Level 1.

You do not need special privileges to override logical protection on a blank disk volume (that is, a volume that has never been written to) or on a disk volume that is owned by your current UIC or by UIC [0,0]. In all other cases, you must have user privilege VOLPRO to initialize a disk volume.

The following example initializes the volume on DMA1 and labels the volume ACCOUNTS:

```
$ INITIALIZE DMA1: ACCOUNTS
```

12.5.4 Initializing Magnetic Tape Volumes

The default format for magnetic tape volumes produced by the INITIALIZE command is based on Level 3 of the American National Standards Institute (ANSI) X3.27-87 and International Standards Organization (ISO) standards for magnetic tape labels and file structure for informational interchange.

If you use ANSI ‘‘a’’ characters (which are not alphanumeric) on the volume label on magnetic tape, you must enclose the volume name in quotation marks.

The following example initializes the volume on MTB1 and labels the volume SOURCE:

```
$ INITIALIZE MTB1: SOURCE
```

12.6 Mounting Volumes

12.6.1 Overview

Before you attempt to use files or data on an allocated disk or tape volume, make sure the volume is mounted. The DCL command MOUNT makes a volume and the files or data it contains accessible to your process.

12.6.2 Using the MOUNT Command

When you enter the MOUNT command, the system verifies that the following conditions have been met:

- The device has not been allocated by another user.
- The device protection allows you to allocate the device.
- A volume is physically loaded on the specified device.
- The label on the volume matches the label you specified.

12.6.3 Volume Sets

You can mount a single volume or a volume set. Binding volumes into a volume set allows you to extend the space available for your files by adding volumes to the same set, rather than by defining multiple, new volumes. The procedures for creating and mounting volume sets (as opposed to single volumes) are described in the *OpenVMS System Manager's Manual*.

12.6.4 MOUNT Command Format

The MOUNT command format is as follows:

```
MOUNT device-name[:][,...] [volume-label[,...]] [logical-name[:]]
```

The elements are as follows:

device-name Specifies the physical device name or logical name of the device on which the volume is to be mounted.

volume-label Specifies the label with which the volume was initialized. You do not need to specify the volume label if you use one of the following MOUNT qualifiers: /FOREIGN, /NOLABEL, or /OVERRIDE=IDENTIFICATION.

logical-name Defines a name to be associated with the device. If you omit the logical name, the MOUNT command assigns the default logical names DISK$volume-label and TAPE$volume-label to disk and tape drives, respectively.

12.6.5 Requesting Operator Assistance

Operators can perform the physical mounting (and dismounting) of both system and private volumes. If no operator is available (operator is not enabled) to receive and respond to a MOUNT request, a message is displayed to inform you of the situation. A volume placed in the requested drive needs no additional operator assistance. Note that you can specify the /NOASSIST qualifier to avoid operator assistance.

MOUNT messages are sent to all operators enabled to receive TAPE and DISK messages. Thus, if operator assistance is needed for mounting a disk device, a message is sent to disk operators.

12.6.6 Example: Requesting Operator Assistance

The MOUNT command shown here notifies the operator of your mount request and displays a message at your terminal:

```
$ MOUNT DMA1: DISK VOL1
%MOUNT-I-OPRQST, PLEASE MOUNT DEVICE _MARS$DMA1:
```

After the device has been successfully mounted, you are notified with the following message:

```
%MOUNT-I-MOUNTED, DISK mounted on _DMA1:
```

12.6.7 Example: Mounting Disk Volumes

The following example shows how to allocate, initialize, and mount a disk volume:

```
$ ALLOCATE DMA2:  TEMP
%DCL-I-ALLOC, _MARS$DMA2: allocated
$ INITIALIZE  TEMP:  BACKUP_FILE
$ MOUNT  TEMP:  BACKUP_FILE
%MOUNT-I-MOUNTED, BACKUP_FILE mounted on _DMA2:
$ CREATE/DIRECTORY  TEMP:[ARCHIE]
```

Before you can place any files on the volume, you must create a directory, as shown by the CREATE/DIRECTORY command.

12.6.8 Mounting a Foreign Disk Volume

To mount a foreign disk volume (that is, one having a file structure other than Files–11), use the /FOREIGN qualifier. For example:

```
$ MOUNT/FOREIGN DISK
%MOUNT-I-MOUNTED, BACKUP_FILE      mounted on DISK$DMA2:
```

The MOUNT/FOREIGN command makes the contents of your volume available to the system but makes no assumptions concerning its file structure. In the preceding example, MOUNT reports a volume label, indicating that the disk has a Files–11 structure, even though it was mounted as a foreign device. If a disk does not have a recognized file structure, MOUNT does not display a label.

Note that you need the user privilege VOLPRO to mount a Files–11 structured disk with the /FOREIGN qualifier, unless its owner UIC matches your own.

12.6.9 Mounting Magnetic Tape Volumes

When you use the MOUNT command to mount a magnetic tape volume, the system checks to see whether the volume has an ANSI-labeled format. If the format is ANSI-labeled, MOUNT checks the following:

- The volume identifier field
- The protection on the ANSI-labeled volume

12.6.10 Example

The command shown in this example mounts an ANSI-labeled volume and assigns a logical name to it:

```
$ MOUNT  MT:  SYSTPV ET
%MOUNT-I-OPRQST, please mount volume SYSTPV in device $MTA1:
%MOUNT-I-MOUNTED, SYSTPV mounted on MTA1:
%MOUNT-I-RQSTDON, operator request canceled -- mount completed successfully
```

MOUNT finds an available MT drive, MTA1, and requests operator assistance. The message displayed indicates which drive has been selected. At this point, you (or the operator) load the magnetic tape on the drive and the mount operation completes. No operator response is necessary. The display informs you that the volume named SYSTPV is mounted on the drive MTA1. Although MOUNT does not require a logical name, this example assigns the logical name ET to the volume SYSTPV.

12.7 Accessing Files on Private Devices

12.7.1 Overview

To access a file that is on a private device, you must either specify the device name or use the SET DEFAULT command to set default to the device. For files on disks, you must also specify the directory name.

You can use physical, logical, or generic names described in the following sections to refer to devices. In addition, if your system is part of a VMScluster, certain devices are accessible to all members of the VMScluster. Section 12.7.8 describes the syntax for VMScluster device names.

12.7.2 Using Physical Device Names

Each physical device known to the system is uniquely identified by a **physical device name**. The physical device name identifies the kind of device; for example, a storage disk or a terminal.

Most physical device names consist of:

- A device code, which represents the hardware device type
- A controller designator, which identifies the hardware controller to which the device is attached
- A unit number, which identifies a device on a particular controller.

For information on specific device-naming formats, see *OpenVMS System Manager's Manual*.

12.7.3 Accessing Files on Volume Sets

When a disk or tape is mounted on a device, the system recognizes it as a volume. The system also recognizes volume sets. A **volume set** consists of two or more related volumes. To access a file on a tape volume set, specify any device that has been allocated to it.

To access a file on a disk volume set, you have the following options:

- Specify the name of the device on which the first volume in the set is mounted.
- Specify the logical name that was assigned to the volume set when it was mounted.

If you do not specify a device name, the system supplies the current default device name. For more information on volumes and volume sets, see the *OpenVMS System Manager's Manual*.

12.7.4 Using Device Names with Commands

Some commands accept output file specifications. You can replace an output file specification with the name of a record-oriented device such as a printer or a terminal. For example:

```
$ COPY DISFILE.DAT TTB4:
```

The COPY command sends the file DISFILE.DAT to the terminal named TTB4. The terminal accepts and displays the file one record at a time. When you use a device name as a file specification, follow the device name with a colon (:).

12.7.5 Using Logical Device Names

Your system manager can set up **logical device names** to represent the devices on your system. Logical device names equate a somewhat cryptic device name to a short, meaningful name. You can use these logical device names, rather than the physical device names, to refer to devices.

When you use a logical device name in a file specification, follow it with a colon.

Chapter 13 describes in more detail how to use logical names.

12.7.6 Example

In the following example, COD1 is a logical device name for the device that contains the disk volume with the file [NOAH]ANIMALS.LIS:

```
$ TYPE COD1:[NOAH]ANIMALS.LIS
```

As long as the system manager defines the logical name COD1 correctly, the system can access the file, regardless of where the volume is mounted.

12.7.7 Using Generic Device Names

A **generic device name** consists of the device code and omits the specific controller or unit number. When you use a generic device name with the MOUNT or ALLOCATE commands, the system locates the first available controller or device unit whose physical name satisfies the portions of the generic device name you specified.

If you specify a generic device name for any other command, the following defaults apply:

- If you omit the controller designation, it is assumed to be A.
- If you omit the unit number, it is assumed to be 0.

12.7.8 Using VMScluster Device Names

A VMScluster device name includes the name of the node to which the device is attached and the physical device name, separated by a dollar sign ($). For example, ROXXY$DUA1 refers to disk DUA1 on node ROXXY.

As a general rule, always use an allocation class device name to identify dual-pathed VMScluster disks. It is the only name that all VMScluster nodes recognize at all times.

For more information on the device name format in VAXcluster or VMScluster environments, see *VMScluster Systems for OpenVMS*.

12.7.9 VMScluster Device Name Format

If a device is dual pathed (connected to two nodes), specify the VMScluster device name in the following format:

$allocation-class$ddcu

The elements are:

allocation-class	A value assigned to the nodes connecting a dual-pathed device. For example, 1DJA16 identifies a disk that is dual pathed between two nodes that both have an allocation class value of 1.
dd	Represents device code of the hardware device type (for example, the device code DK represents an RZ23 disk).
c	Identifies the hardware controller to which the device is attached. The controller designation, along with the unit number, identifies the location of the device within the hardware configuration of the system. Controllers are designated with alphabetic letters A to Z.
u	Uniquely identifies the unit number of a device on a particular controller. Unit numbers are decimal numbers from 0 to 65535.

12.8 Dismounting Volumes

12.8.1 Overview

When you are done with the files on a disk or tape volume, you can use the DISMOUNT command to dismount the volume. Before a volume is dismounted, the DISMOUNT command checks for conditions that could prevent the dismount from completing. For example, if the volume contains installed swap and page files, installed images, or open user files, DISMOUNT displays an error message indicating that the volume cannot be dismounted.

12.8.2 Logical Dismounting

By default, the DISMOUNT command automatically unloads the volume from the drive. If you plan to mount or initialize a volume again after you dismount it, you can save time and eliminate unnecessary handling of that volume by using the /NOUNLOAD qualifier. For example:

```
$  DISMOUNT/NOUNLOAD MTA1:
```

In this example, the magnetic tape volume is logically dismounted and the tape is rewound but the tape remains physically loaded on drive MTA1.

12.8.3 Dismounted Volumes

You should always explicitly dismount a volume with the DISMOUNT command before physically unloading the volume. Wait for the drive to unload before you remove the volume. (You can verify that the dismount is complete by entering the DCL command SHOW DEVICES.)

A volume is dismounted and unloaded automatically if you log out of the job from which you had mounted the volume. If the system fails, however, the volume is not automatically dismounted.

12.8.4 Allocated Devices

If the device you are dismounting was allocated with an ALLOCATE command, it remains allocated after it is dismounted with the DISMOUNT command. If the device was implicitly allocated by the MOUNT command, the DISMOUNT command deallocates it.

13

Logical Names: Defining Names for Devices and Files

13.1 Overview

A logical name is a name that can be used in place of another character string to represent system objects such as files, directories, devices, or queues. This chapter includes information about the following:

- Logical name usage
- Using system-defined logical names
- Creating logical names
- Creating logical name tables
- Process directory table LNM$PROCESS_DIRECTORY
- System directory table LNM$SYSTEM_DIRECTORY
- Displaying logical names
- Deleting logical names and logical name tables
- Redefining process-permanent logical names
- Using process-permanent logical names as file specifications
- Logical name translation
- Search lists

13.1.1 References

For additional information on the commands described in this chapter, refer to the *OpenVMS DCL Dictionary* or online help.

13.2 About Logical Names

13.2.1 Usage

Logical names:

- Are equated to strings (called equivalence strings or equivalence names) or a list of equivalence strings (called search lists). When you use a logical name, the equivalence string is substituted for the logical name.
- Can be shorthand for long file specifications.
- Can be defined by you or by the system.

- Can be used to keep programs and command procedures independent of physical file specifications. For example, if a command procedure references the logical name ACCOUNTS, you can equate ACCOUNTS to any file on any disk before executing the command procedure.

In general, when a command accepts a system object, the command checks whether the name you provide is a logical name. If the name is a logical name, the system replaces the logical name with its actual value and executes the command.

13.2.2 Example

In the following example, the logical name COMS is created to represent the directory DISK7:[WALSH.COMMAND_PROC]:

```
$ TYPE COMS:PAYROLL.COM
```

Then COMS can be used in a DCL command line:

```
$ SET DEFAULT COMS
```

13.2.3 Using Logical Names for File Input/Output (I/O)

In command procedures, you can use logical names to perform input and output from files. When you **open** a file with the OPEN command, you also create a logical name for the file. The READ, WRITE, and CLOSE commands use the logical name and not the actual file specification to refer to the file.

13.2.4 Example

In the following example, when the CLOSE command **closes** the file, it deassigns the logical name INFILE:

```
$ OPEN INFILE DISK3:[WALSH]DATA.DAT
$ READ INFILE RECORD
$ CLOSE INFILE
```

13.3 Using System-Defined Logical Names

13.3.1 Systemwide Logical Names

The system creates a set of systemwide logical names for you when you start the system and log in. These logical names allow you to refer to commonly used files or devices without using their physical device names.

Every time you log in, the system creates a group of logical names for your process and places these names in your process table.

13.3.2 Examples

- To list your operating system's programs, you do not need to know the name of the disk and directory where these programs are stored. Instead, you can use the logical name SYS$SYSTEM, as follows:

  ```
  $ DIRECTORY SYS$SYSTEM
  ```

- The logical name SYS$LOGIN refers to your default device and directory when you log in. If you have changed your current defaults by using the SET DEFAULT command, you can use the following command to display a file from your initial default directory:

  ```
  $ TYPE SYS$LOGIN:DAILY_NOTES.DAT
  ```

13.4 Creating Logical Names

13.4.1 Overview

You can create logical names with either the ASSIGN command or the DEFINE command. Usually, you define logical names in a login command procedure (LOGIN.COM), so you can use the logical name each time you log in. You can also create logical names interactively. However, you can use these logical names only while your current process is active. Your system manager can create logical names for use by anyone logged in to the system.

13.4.2 Using the DEFINE Command

The format for defining a logical name with the DEFINE command is as follows:

DEFINE logical-name equivalence-string[,...]

The same format can be used to create logical names for other system objects.

By default, the DEFINE command places logical names in your process logical name table, where the logical name is available only to your process and subprocesses. Section 13.5 describes logical name tables.

13.4.3 Examples

- In the following example, the following command creates the logical name WORKFILE and equates it the equivalence string DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT:

  ```
  $ DEFINE WORKFILE DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT
  ```

  After you define WORKFILE as a logical name, you can use the logical name interchangeably with the equivalence string.

- In the following example, the command creates the logical name MY_Q for the print queue BLDGC_LPS20_ANSI:

  ```
  $ DEFINE MY_Q BLDGC_LPS20_ANSI
  ```

  You could then use the following command to print the file FABLES.TXT to the BLDGC_LPS20_ANSI print queue:

  ```
  $ PRINT/QUEUE=MY_Q FABLES.TXT
  ```

13.4.4 Rules for Creating Logical Names

Observe the following rules when you create a logical name with the DEFINE command:

- Limit the equivalence string and the logical name to no more than 255 characters each. A logical name can contain alphanumeric characters, the underscore (_), the dollar sign ($), and the hyphen (-).
- When you specify an equivalence string, include the punctuation marks (colons, brackets, periods) required by a file specification. For example, end a device name with a colon, enclose a directory specification in brackets, and precede a file type with a period.
- If a logical name represents only part of a file specification, separate the name from the rest of the file specification with a colon. When you use a logical name to represent a complete file specification, the terminating colon is not needed.

  In addition, be sure the logical name is the leftmost component of a file specification.
- Optionally, end the logical name with a colon.

  Note that the ASSIGN command removes the colon before placing the logical name in a logical name table; the DEFINE command saves the colon as part of the logical name.
- If you equate a logical name to one equivalence string, then equate the same logical name to a different equivalence string, the second definition will supersede the first, unless you define them in different **logical name tables** or define them with different access modes.

13.4.5 Example

The following commands display the file DISK1:[SALES_STAFF]PAYROLL.DAT:

```
$ DEFINE PAY DISK1:[SALES_STAFF]PAYROLL.DAT
$ TYPE PAY

$ DEFINE PAY_FILE DISK1:[SALES_STAFF]PAYROLL
$ TYPE PAY_FILE:*.DAT

$ DEFINE PAY_DIR DISK1:[SALES_STAFF]
$ TYPE PAY_DIR:PAYROLL.DAT

$ DEFINE PAY_DISK DISK1:
$ TYPE PAY_DISK:[SALES_STAFF]PAYROLL.DAT
```

13.4.6 Creating Logical Node Names

You can use a logical node name in place of a network node name or in place of a node name and an access control string. Once you define a logical node name, you can use it to avoid typing (and displaying) your user name and password on the screen.

13.4.7 Rules for Creating Logical Node Names

To define a logical node name, observe the following rules:

- Do not begin the logical name with an underscore (_).
- End the equivalence string with a double colon (::) and enclose it in quotation marks (" ").
- Use two sets of quotation marks ("" "") where you want quotation marks to appear in the access control string.
- Specify a logical name that contains 1 to 255 characters.

Caution

Do not place a DEFINE command that includes a password in a file (your login command procedure, for example). If others read the file, they will see the password.

13.4.8 Example

In the following example, the command equates the logical name BOS to the node name BOSTON and an access control string, where ADAMS is the user name and OLMEKIKA is the password:

```
$ DEFINE BOS "BOSTON""ADAMS OLMEKIKA""::"
```

13.4.9 Using Logical Node Names in File Specifications

A file specification can contain both a logical node name (which the system translates at the local node) and a logical device name (which the system translates at the remote node). If you use a logical name to represent a node name only, you must include a double colon (::) when you use the logical name in the node position of a file specification.

After the system translates a logical node name at the local node, it parses the rest of the file specification to determine whether the format is valid.

13.4.10 Example

In the following example, the system translates the logical node name NYC at the local node. It translates the device name (DOC:) at the remote node (NEWYRK):

```
$ DEFINE NYC NEWYRK::
$ TYPE NYC::DOC:[PERKINS]TERM_PAPER.DAT
```

13.4.11 Overriding Access Control Strings

To override the access control string in a logical node name, specify both the logical name and an access control string in the command line.

When the system translates a logical node name iteratively, the access control information in the logical node name that is translated first overrides following access control information.

13.4.12 Examples

- In the following example, the access control string "REVERE HTEBAZILE" overrides the access control string given in the equivalence string for BOS:

```
$ DEFINE BOS "BOSTON""ADAMS OLMEKIKA"""::"
$ TYPE BOS"REVERE HTEBAZILE"::RIDE.DAT
```

- In the following example, the logical name TEST1 translates to TORONTO"TEST NAMWENLUAP"::DBA1: :

```
$ DEFINE TORONTO "TRNTO""TEST EIZNEKCAM"""::"
$ DEFINE TEST1 "TORONTO""TEST NAMWENLUAP"""::DBA1:"
$ TYPE TEST1:PROC.DAT
```

  TORONTO is a logical node name, so **iterative translation** occurs. In other words, the operating system searches the logical name tables until all levels of logical names in a definition are found. However, the access control string in the DEFINE TEST1 logical name assignment overrides the access control string in the DEFINE TORONTO logical node name assignment. Therefore, the TYPE command displays the following file:

```
TRNTO"TEST NAMWENLUAP"::DBA1:PROC.DAT
```

13.4.13 Translation Attributes

When you create a logical name, you can specify translation attributes that modify how the system interprets the equivalence string.

To apply translation attributes to an equivalence string, use the /TRANSLATION_ATTRIBUTES qualifier to the DEFINE command. This is a positional qualifier: depending on where you place it on a command line, it can apply translation attributes to all equivalence strings or only to certain ones.

13.4.14 Creating Concealed Logical Names

The CONCEALED attribute causes system messages to display the logical name rather than the physical name of a device. Typically, you use the CONCEALED attribute with logical names that represent physical devices. Using concealed devices lets you write programs, write command procedures, and perform other operations without being concerned about which physical device holds the disk or tape. It also lets you use names that are more meaningful than the physical device names.

13.4.15 Creating Terminal Logical Names

The TERMINAL attribute prevents iterative translation of a logical name (that is, the equivalence string is not examined to see if it is also a logical name). The translation is ‘‘terminal’’ (final or completed) after the first translation.

13.4.16 Example: Translation Attributes

In the following example, the device name DJA3: is concealed with the logical name DISK:

```
$ DEFINE/TRANSLATION_ATTRIBUTES=CONCEALED DISK DJA3:
$ SHOW DEFAULT
  DISK:[SAM.PUP]
$ SHOW LOGICAL DISK
  "DISK" = "DJA3" (LNM$PROCESS_TABLE)
```

The logical name DISK represents the physical device DJA3. Thus, the SHOW DEFAULT command displays the logical name DISK rather than the physical device name, DJA3. The SHOW LOGICAL command reveals the translation of DISK.

13.4.17 Equating Several Logical Names to One Equivalence String

You can equate more than one logical name to an equivalence string by entering more than one DEFINE command. For example, the following commands equate the logical names $TERMINAL and CONSOLE to the physical name of a terminal so that both logical names translate to the same device (LTA69):

```
$  DEFINE $TERMINAL LTA69
$  DEFINE CONSOLE LTA69
```

13.4.18 Search Lists

To create a search list, equate a logical name to more than one equivalence string in a single command.

When you use a search list, the system translates the logical name until it finds a match. The order in which you specify the equivalence strings determines the order in which the system translates the names. It uses each equivalence string listed in the definition until a match is found. Note that a search list is not a wildcard; it is a list of places to look. Once a file is found, the search ends.

13.4.19 Examples

- In the following example, the logical name GETTYSBURG is a search list:

  ```
  $ DEFINE GETTYSBURG [JONES.HISTORY],[JONES.WORKFILES]
  $ SHOW LOGICAL GETTYSBURG

    "GETTYSBURG" = "[JONES.HISTORY]" (LNM$PROCESS_TABLE)
         = "[JONES.WORKFILES]"
  ```

- In the following example, the TYPE command searches the equivalence string [JONES.HISTORY] before it searches [JONES.WORKFILES] (the order specified in the preceding logical name definition for GETTYSBURG):

  ```
  $ TYPE GETTYSBURG:SPEECH.TXT

  DISK1:[JONES.HISTORY]SPEECH.TXT;2
  ```

```
Fourscore and seven years ago, our fathers brought
forth on this continent a new nation, conceived
in liberty, and dedicated to the proposition that
all men are created equal.
   .
   .
   .
```

Once the TYPE command finds a file named SPEECH.TXT, it ends the search and displays the file.

13.4.20 Using Search Lists with Wildcards

You can use a search list with a command that accepts wildcards. When you use wildcards, the system forms file specifications by using each equivalence string in the search list. The command operates on each file specification that identifies an existing file.

13.4.21 Example

In the following example, the DIRECTORY command is specified with a wildcard character in the version field. It finds all versions of SPEECH.TXT in the search list defined by GETTYSBURG:

```
$ DIRECTORY GETTYSBURG:SPEECH.TXT;*

Directory DISK1:[JONES.HISTORY]

SPEECH.TXT;2     SPEECH.TXT;1

Total of 2 files.

Directory DISK1:[JONES.WORKFILES]

SPEECH.TXT;1

Total of 1 file.

Grand total of 2 directories, 3 files.
```

13.4.22 Imprecise File Specifications

When you enter a search list (for example, using the DIRECTORY command), the operating system uses elements in one part of the list to supply parts of the file specification that are omitted from other parts of the list. If a file specification is not precise (as shown in the following example), command lines can produce multiple files and file-not-found conditions:

```
$ DIRECTORY SYS$MANAGER:LOGIN.COM,SYS$LOGIN
```

You can avoid producing multiple files and file-not-found conditions by placing a semicolon after the file specification, as shown:

```
$ DIRECTORY SYS$MANAGER:LOGIN.COM;,SYS$LOGIN
```

13.4.23 Logical Name Tables

Identical logical names can exist in more than one logical name table (see Section 13.5). The logical name that the system uses depends on the order in which it searches the logical name tables. For example, when the system attempts to translate a logical name to identify the location of a file, it uses the logical name LNM$FILE_DEV to provide the list of tables in which to look for the name. The order in which the tables are listed is also the order in which they are searched.

By default, the precedence order defined by LNM$FILE_DEV is:

1. Process table
2. Job table
3. Group table
4. System table

So, for example, if a logical name exists in both the process and the group logical name tables, the logical name within the process table is used. See Section 13.5.10 for information about redefining LNM$FILE_DEV.

13.4.24 Using Different Logical Name Tables

By default, the DEFINE and DEASSIGN commands place names in, and delete names from, your process table. However, you can request a different table with the /TABLE qualifier. For example:

```
$ DEFINE/TABLE=LNM$SYSTEM REVIEWERS DISK3:[PUBLIC]REVIEWERS.DIS
```

This command creates the logical name REVIEWERS in the system table.

13.4.25 Access Modes

The OpenVMS operating system has four access modes, as follows:

- User mode (the outermost and least privileged mode)
- Supervisor mode
- Executive mode
- Kernel mode (the innermost and most privileged mode)

You can use the DCL commands DEFINE or ASSIGN to create logical names in the first three modes (user, supervisor, and executive). By specifying different access modes for each logical name definition, you can equate the same logical name to different equivalence strings in the same logical name table. Note that you must have SYSNAM or SYSPRV privileges to create logical names in executive mode in any logical name table.

When you use the DEFINE command without specifying a mode, DCL creates the logical name in supervisor mode.

13.4.26 Example

In the following example, the commands shown equate the logical name ACCOUNTS to two different equivalence strings in the process logical name table—one in supervisor mode and one in executive mode:

```
$ DEFINE ACCOUNTS DISK1:[ACCOUNTS]CURRENT.DAT
$ DEFINE/EXECUTIVE_MODE ACCOUNTS DISK1:[JANE.ACCOUNTS]OBSOLETE.DAT
```

13.4.27 User Mode

Logical names created in user mode are temporary. Define a logical name in user mode when you want to use it only for the execution of the next command or image.

13.4.28 Example

In the following example, the logical name ADDRESSES is deleted automatically after the execution of the program PAYABLE:

```
$ DEFINE/USER_MODE ADDRESSES DISK1:[SAM.ACCOUNTS]OVERDUE.LIS
$ RUN PAYABLE
```

13.4.29 Executive Mode

In looking up logical names, all privileged images and utilities, such as LOGINOUT and Mail, bypass the user-mode and supervisor-mode portions of the system logical name table. Therefore, Digital recommends that you define all logical names for important system components (public disks and directories, for example) in executive mode. (Only the operating system and privileged programs can create logical names in kernel mode.)

13.5 Creating Logical Name Tables

13.5.1 Overview

The system stores logical names and their equivalence strings in process, job, group or system logical name tables. Some logical name tables are available only to your process; these tables are called process-private. Other tables are shareable; that is, they are available to other users on the system.

13.5.2 Process Table

The names in this table are available only to your process and any subsequent subprocesses. Every process on the system has a process logical name table. When you log in, the system creates logical names for your process and places them in your process table.

The name for your process table is LNM$PROCESS_TABLE. However, use the logical name LNM$PROCESS to refer to your process table.

13.5.3 Job Table

The names in this table are available to your process and to any of your subprocesses. All job tables are shareable so that all users can access them.

The name for your job table is LNM$JOB_xxx (xxx represents the Job Information Block address defined by the system for your job tree). However, use the logical name LNM$JOB to refer to your job table.

13.5.4 Group Table

The names in this table are available to all users in your group. The name for your group table is LNM$GROUP_xxx (xxx represents a group number.) However, use the logical name LNM$GROUP to refer to your group table.

13.5.5 System Table

The names in this table are available to all users on the system. The name for your system table is LNM$SYSTEM_TABLE. However, use the logical name LNM$SYSTEM to refer to the system table.

13.5.6 Creating Logical Names in Tables

In general, you create logical names in your process table. However, if you are a system manager or if your work requires you to add names to shareable tables (see Section 13.5.9), you must also work with the group and system tables. To create or delete a name in your group table, you need GRPNAM, GRPPRV, or SYSPRV privilege. To create or delete a name in the system table, you must have a UIC group number between 0 and 10 or SYSNAM or SYSPRV privilege.

The CREATE/NAME_TABLE command creates a logical name table and catalogs it in one of the directory logical name tables. Logical names that identify logical name tables or that translate iteratively to logical name tables must always be entered into one of the directory logical name tables.

13.5.7 Process Private Logical Name Tables

To create a logical name table that is private to your process, create the table in LNM$PROCESS_DIRECTORY (the default). If the table needs to be shareable, specify /PARENT_TABLE=LNM$SYSTEM_DIRECTORY with the CREATE/NAME_TABLE command.

A name in a directory table can contain 1 to 31 characters. Only alphanumeric characters, the dollar sign ($), and the underscore (_) are valid.

13.5.8 Example

The following example creates a process-private logical name table named TAX, places the definition for the logical name CREDIT in the table, and verifies the table's creation. The SHOW LOGICAL/TABLE command displays logical names in tables other than LNM$SYSTEM or LNM$PROCESS:

```
$ CREATE/NAME_TABLE TAX
$ DEFINE/TABLE=TAX CREDIT [ACCOUNTS.CURRENT]CREDIT.DAT
$ SHOW LOGICAL/TABLE=TAX CREDIT
 "CREDIT" = "[ACCOUNTS.CURRENT]CREDIT.DAT"  (TAX)
```

13.5.9 Shareable Logical Name Tables

To create a shareable logical name table, use the /PARENT_TABLE qualifier and specify the name of a shareable table. For example:

```
$ CREATE/NAME_TABLE/PARENT_TABLE=LNM$SYSTEM_DIRECTORY NEWTAB
```

The following privilege and access requirements apply to the use of shareable logical name tables:

- To create a shareable logical name table, you must have SYSPRV privilege and EXECUTE (E) access to the *parent* table.
- To delete a shareable logical name table, you must have SYSPRV privilege or DELETE (D) access to the table.
- To place a name in or delete a name from a shareable logical name table, you must have WRITE (W) access to the table.
- To read (translate) a name in a shareable logical name table, you must have READ (R) access to the table.

13.5.10 Adding Logical Names to Logical Name Tables

Generally, you do not need to change the default logical name table definitions set up in the directory tables LNM$PROCESS_DIRECTORY and LNM$SYSTEM_DIRECTORY. Two reasons for changing the entries in the directory logical name tables are:

1. To create another logical name table
2. To change the order in which the system searches the logical name tables

To place a logical name in a specific table, use the DEFINE /TABLE command.

13.5.11 Example

In the following example, the logical name TESTDIR is placed in NEWTAB (a user-defined logical name table):

```
$ DEFINE/TABLE=NEWTAB TESTDIR [JONES.TESTFILES]
```

The next time TESTDIR is referred to in a file specification, the system looks for it in the process, job, group, and system logical name tables only. It does not look in the NEWTAB table. NEWTAB must be included in the list of logical name tables that the system searches.

13.5.12 Adding Logical Names to the Logical Name Directory

There are two ways to add logical names to the logical name directory:

- Create a private definition of LNM$FILE_DEV within the process logical name directory table.
- Redefine LNM$FILE_DEV within the system directory logical name table (requires SYSNAM or SYSPRV privilege).

To add a logical name table to the process logical name directory table, redefine the logical name LNM$FILE_DEV within the process logical name directory table.

If you have SYSNAM or SYSPRV privileges, you can add a logical name table to the system directory logical name table.

13.5.13 Examples

- In the following example, NEWTAB is added to the process logical name directory:

  ```
  $ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
  _$ NEWTAB, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
  ```

  This example specifies that the system search the NEWTAB table first for the following reasons:

  – The process-private version of LNM$FILE_DEV is used before the default system version.

  – Within LNM$FILE_DEV, NEWTAB is listed before the process, job, group, and system logical name tables.

- In the following example, the logical name NEWTAB is added to the system directory table by a user with SYSNAM or SYSPRV privileges:

  ```
  $ DEFINE/TABLE=LNM$SYSTEM_DIRECTORY LNM$FILE_DEV -
  _$ NEWTAB, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
  ```

13.5.14 Defining the Protection for a Logical Name Table

There are two ways to define the protection of a logical name table:

- Use the /PROTECTION qualifier with the CREATE/NAME_TABLE command. This command lets you set UIC-based protection for a shareable logical name table.
- Apply ACL protection with the ACL editor or with the SET SECURITY/ACL/OBJECT_TYPE=LOGICAL_NAME_TABLE command. ACLs for system logical name tables are saved when the system reboots but ACLs for process logical name tables are not. You must reestablish ACLs on process logical name tables every time the system is booted. For more information, see the SET SECURITY/ACL command in the *OpenVMS DCL Dictionary*.

13.5.15 Quotas for Logical Name Tables

Quotas are used to limit the amount of system resources that a given logical name table can consume. The process, group, and system logical name tables have an infinite quota. By default, when you create a logical name table, it too has an unlimited quota.

13.5.16 Specifying Quotas

You can specify a quota to limit the size, in bytes, of a logical name table that you create. Before a logical name is created, the size of its data structure is checked against the quota remaining for the table. If there is insufficient quota available for the new entry, the system displays an error message.

Once you set the quota for a table, you cannot change it. If the table runs out of room, use the DEASSIGN command to delete old logical names. This frees space for your new logical names.

13.5.17 Example

In the following example, the logical name table ABC is created and is given a quota of 500 bytes:

```
$ CREATE/NAME_TABLE/QUOTA=500 ABC
```

13.5.18 Setting Job Table Quotas

The job logical name table is a shareable table. The quota for a job logical name table is established when the table is created. The quota is determined by one or more of the following criteria:

- The JTQUOTA value established for the user in the system user authorization file SYSUAF.DAT (if the first image activated by the process was the system image LOGINOUT).
- The PQL$_JTQUOTA quota list value specified in the call to the Create Process ($CREPRC) system service.
- The /JOB_TABLE_QUOTA qualifier value on the RUN command used to create the detached process.
- The SYSGEN parameter PQL_DJTQUOTA (if none of the preceding conditions applies). The standard default value for this parameter is 1024 bytes; however, the system manager can change it. The System Generation utility (SYSGEN) can be used to display and set the values of the parameters PQL_DJTQUOTA (default job logical name table quota) and PQL_MJTQUOTA (minimum job logical name table quota).

A quota value of 0 for a job logical name table means there is no quota. For all practical purposes, the quota is unlimited.

13.5.19 Using Logical Name Directory Tables

When you enter a logical name as part of a command line, the system translates the logical name by searching the logical name tables in a certain order. The system stores information about existing logical name tables and the order in which they are searched in two logical name directory tables:

- LNM$PROCESS_DIRECTORY, which is a process-private directory

- LNM$SYSTEM_DIRECTORY, which is a shareable directory

There is only one shareable directory for each system but each process has its own process-private directory. Both directories contain logical names that translate iteratively to table names. All logical name tables and any logical names that translate to tables are kept in these directories.

13.6 Process Directory Table LNM$PROCESS_DIRECTORY

13.6.1 Overview

Within each logical name table, the system defines some logical names for you. This section describes each table and its system-defined logical names in the process directory table LNM$PROCESS_DIRECTORY. Note that the logical names SYS$INPUT, SYS$OUTPUT, SYS$ERROR, and SYS$COMMAND refer to process-permanent files (files that remain open for the life of the process). For more information on process-permanent files, see Section 13.10.)

13.6.2 LNM$GROUP

LNM$GROUP is a logical name that is defined as LNM$GROUP_ xxx, where xxx represents your group number. LNM$GROUP_ xxx is the logical name table used by your UIC group. The table LNM$GROUP_xxx is cataloged in the system directory table. Therefore, LNM$GROUP is a logical name that translates iteratively to the name of your group logical name table.

13.6.3 LNM$JOB

LNM$JOB is a logical name that is defined as LNM$JOB_ xxx, where xxx represents a number unique to your job tree. LNM$JOB_xxx is the logical name table used by your job. The table LNM$JOB_xxx is cataloged in the system directory table. Therefore, LNM$JOB is a logical name that translates iteratively to the name of your job logical name table.

13.6.4 LNM$PROCESS

LNM$PROCESS is a logical name that translates iteratively to LNM$PROCESS_TABLE, which is the name of your process logical name table.

13.6.5 LNM$PROCESS_DIRECTORY

LNM$PROCESS_DIRECTORY is the name of your process directory logical name table.

13.6.6 LNM$PROCESS_TABLE (Process Table)

LNM$PROCESS_TABLE is the name of your process logical name table, which is available only to your process and any subsequent subprocesses. By default, the table contains the following logical names:

- SYS$COMMAND

  The initial file (usually your terminal) from which DCL reads input. A file from which DCL reads input is called an input stream. The command interpreter uses SYS$COMMAND to ''remember'' the original input stream.

- SYS$DISK

  The default device established at login or changed by the SET DEFAULT command.

- SYS$ERROR

  The default device or file to which DCL writes system error messages generated by warnings, errors, and severe errors.

- SYS$INPUT

  The default file from which DCL reads input.

- SYS$NET

  The source process that invokes a target process in DECnet for OpenVMS task-to-task communication. When opened by the target process, SYS$NET represents the logical link over which that process can exchange data with its partner. SYS$NET is defined only during task-to-task communication.

- SYS$OUTPUT

  The default file (usually your terminal) to which DCL writes output. A file to which DCL writes output is called an output stream.

- TT

  The default device name for terminals.

13.6.7 LNM$JOB_xxx (Job Table)

The job table contains logical names that are available to all processes in your job tree. There is one job table for each job tree in the system.

The system places logical names created for mounted disks, mounted tapes, and temporary mailboxes in the job logical name table. In addition, the system creates the following logical names:

- SYS$LOGIN

  Your default device and directory when you log in.

- SYS$LOGIN_DEVICE

  Your default device when you log in.

- SYS$REM_ID

  For jobs initiated through a DECnet for OpenVMS network connection, the identification of the process on the remote node from which the job was originated. On an OpenVMS operating system, if proxy logins are enabled, this identification is the process's user name; or if proxy logins are not enabled, this is the process identification (PID) number. (Proxy logins to proxy accounts allow users to access files across the network without specifying an access control string.)

- SYS$REM_NODE

  For jobs initiated through a DECnet for OpenVMS network connection, the name of the remote node from which the job was originated.

- SYS$SCRATCH

  Default device and directory to which temporary files are written.

13.7 System Directory Table LNM$SYSTEM_DIRECTORY

This section describes each table and its system-defined logical names in the system directory table LNM$SYSTEM_ DIRECTORY.

13.7.1 LNM$DCL_LOGICAL

LNM$DCL_LOGICAL is a logical name defined as LNM$FILE_ DEV. Translates iteratively into the list of logical name tables searched and displayed by the SHOW LOGICAL command, the SHOW TRANSLATION command, and the F$TRNLNM lexical function. By default, these commands search and display the process, job, group, and system logical name tables, in that order.

13.7.2 LNM$DIRECTORIES

LNM$DIRECTORIES is a logical name defined as LNM$PROCESS_DIRECTORY and LNM$SYSTEM_DIRECTORY.

13.7.3 LNM$FILE_DEV

LNM$FILE_DEV is a logical name defined as the list of logical name tables searched by the system when processing a file specification. Defined as LNM$PROCESS, LNM$JOB, LNM$GROUP, and LNM$SYSTEM so the system searches the process, job, group, and system logical name tables, in that order.

13.7.4 LNM$GROUP_xxx (Group Table)

The group table contains logical names that are available to all users with the same user identification code (UIC) group number. The group table is named LNM$GROUP_xxx, where xxx represents your UIC group number. Use the logical name LNM$GROUP to refer to your group table. Every group on the system has a corresponding group logical name table.

13.7.5 LNM$JOB_xxx

LNM$JOB_xxx is a job logical name table, where xxx is a number unique to this job tree. There is an LNM$JOB_xxx logical name table for each job in the system.

13.7.6 LNM$PERMANENT_MAILBOX

LNM$PERMANENT_MAILBOX is a logical name that is defined as LNM$SYSTEM. Logical names associated with permanent mailboxes are entered in the logical name table to which the logical name LNM$PERMANENT_MAILBOX iteratively translates.

13.7.7 LNM$SYSTEM

LNM$SYSTEM is a logical name that translates iteratively to LNM$SYSTEM_TABLE.

13.7.8 LNM$SYSTEM_TABLE (System Table)

LNM$SYSTEM_TABLE is the name of the system logical name table. The system table contains logical names that are available to all users of the system at the system level. Use the logical name LNM$SYSTEM to refer to it.

The following logical names are automatically defined in the system table when the system starts up:

- **DBG$INPUT**
  Default input stream for the OpenVMS Debugger, equated to SYS$INPUT at the process level.
- **DBG$OUTPUT**
  Default output stream for the OpenVMS Debugger, equated to SYS$OUTPUT at the process level.
- **SYS$COMMON**
  SYS$SYSDEVICE:[SYS*n*.SYSCOMMON.], where *n* is the root directory number of your processor. Device and directory name for the common part of SYS$SYSROOT.

- **SYS$ERRORLOG**
  Device and directory name of error log data files. By default, SYS$SYSROOT:[SYSERR].
- **SYS$EXAMPLES**
  Device and directory name of system examples. By default, SYS$SYSROOT:[SYSHLP.EXAMPLES].
- **SYS$HELP**
  Device and directory name of system help files. By default, SYS$SYSROOT:[SYSHLP].
- **SYS$INSTRUCTION**
  Device and directory name of system instruction data files. By default, SYS$SYSROOT:[SYSCBI].
- **SYS$LIBRARY**
  Device and directory name of system libraries. By default, SYS$SYSROOT:[SYSLIB].
- **SYS$LOADABLE_IMAGES**
  Device and directory of operating system executive loadable images, device drivers, and other executive-loaded code. By default, SYS$SYSROOT:[SYS$LDR].
- **SYS$MAINTENANCE**
  Device and directory name of system maintenance files. By default, SYS$SYSROOT:[SYSMAINT].
- **SYS$MANAGER**
  Device and directory name of system manager files. By default, SYS$SYSROOT:[SYSMGR].
- **SYS$MESSAGE**
  Device and directory name of system message files. By default, SYS$SYSROOT:[SYSMSG].
- **SYS$NODE**
  Network node name for the local system if DECnet for OpenVMS is active on the system and you are connected to a network.
- **SYS$PROCDMP**
  Directory (set by user) into which image dumps are to be written. No default setting.
- **SYS$SHARE**
  Device and directory name of system shareable images. By default, SYS$SYSROOT:[SYSLIB].
- **SYS$SPECIFIC**
  Device and directory name for node-specific part of SYS$SYSDEVICE. By default, SYS$SYSDEVICE:[SYS*n*.], where *n* is the root directory number of your processor.

- **SYS$STARTUP**
  Device and directory name of system startup files. By default, a search list that points first to SYS$SYSROOT:[SYS$STARTUP] and then to SYS$MANAGER.
- **SYS$SYSDEVICE**
  System disk containing system directories (usually SYS$DISK).
- **SYS$SYSROOT**
  Device and root directory for system directories. By default, a search list that points first to SYS$SYSDEVICE:[SYS*n*.], where *n* is the root directory number of your processor, and then points to SYS$COMMON.
- **SYS$SYSTEM**
  Device and directory of operating system programs and procedures. By default, SYS$SYSROOT:[SYSEXE].
- **SYS$TEST**
  Device and directory name of User Environment Test Package (UETP) files. By default, SYS$SYSROOT:[SYSTEST].
- **SYS$UPDATE**
  Device and directory name of system update files. By default, SYS$SYSROOT:[SYSUPD].

13.7.9 LNM$TEMPORARY_MAILBOX

LNM$TEMPORARY_MAILBOX is a logical name that is defined as LNM$JOB. Logical names associated with temporary mailboxes are entered in the logical name table to which the logical name LNM$TEMPORARY_MAILBOX iteratively translates.

13.8 Displaying Logical Names

13.8.1 Using the SHOW LOGICAL Command

Use the SHOW LOGICAL command to display logical names and their equivalence strings.

Sometimes the definition of a logical name includes another logical name. The SHOW LOGICAL command performs iterative translations. It then displays both the equivalence string and the level of translation. Level numbers are zero based; that is, 0 is the first level, 1 is the second, and so on. To display only the first translation found for a specified logical name, use the SHOW TRANSLATION command. (For more information, see the *OpenVMS DCL Dictionary*.)

13.8.2 Examples

- In the following example, the logical name MYDISK, is displayed. Two translations are performed; the number 1 indicates the second level of translation:

```
$ SHOW LOGICAL MYDISK
  "MYDISK" = "WORK4" (LNM$PROCESS_TABLE)
1 "WORK4" = "$255$DUA17:" (LNM$SYSTEM_TABLE)
```

- In the following example, the equivalence string for the logical name WORKFILE is displayed:

```
$ SHOW LOGICAL WORKFILE
"WORKFILE" = "DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT" (LNM$PROCESS_TABLE)
```

The logical name, its translation, and the table where the logical name is found are displayed.

13.8.3 Displaying Process Permanent Files

If you use the SHOW LOGICAL command to determine the equivalence string for a process-permanent file (see Section 13.10), the command displays only the device portion of the string. For example:

```
$ SHOW LOGICAL SYS$INPUT
"SYS$INPUT" = "_TTB4:" (LNM$PROCESS_TABLE)
```

13.8.4 Displaying the Access Mode for a Logical Name

To display the access mode for a logical name, use the SHOW LOGICAL/FULL command, as follows:

```
$ SHOW LOGICAL/FULL PROJECT
  "PROJECT" [super] = "DISK1:[JONES]" (LNM$PROCESS_TABLE)
```

This example displays the logical name PROJECT in supervisor mode.

13.8.5 Displaying Logical Name Tables

By default, the SHOW LOGICAL command searches your process, job, group, and system tables. However, you can change the default search order or specify other logical name tables to be searched. To display the entries in a particular logical name table, use the /TABLE qualifier. You can also use the /GROUP, /SYSTEM, /JOB, and /PROCESS qualifiers to display the logical names in the group, system, job, and process logical name tables, respectively.

13.8.6 Example

In the following example, the command displays the logical names in the process logical name table (LNM$PROCESS):

```
$ SHOW LOGICAL/TABLE=LNM$PROCESS
(LNM$PROCESS_TABLE)
  "DECW$DISPLAY" = "_WSA30:"
  "SYS$COMMAND" = "_FIFI$VTA65:"
  "SYS$DISK" [super] = "WORK1:"
  "SYS$DISK" [exec] = "WORK1:"
  "SYS$ERROR" = "_FIFI$VTA65:"
  "SYS$INPUT" = "_FIFI$VTA65:"
  "SYS$OUTPUT" [super] = "_FIFI$VTA65:"
  "SYS$OUTPUT" [exec] = "_FIFI$VTA65:"
  "TT" = "_VTA65:"
```

13.8.7 Displaying Directory Table Structure

To display the relationship of directory tables to logical name tables, enter the SHOW LOGICAL/STRUCTURE command, as shown in the following example:

```
$ SHOW LOGICAL/STRUCTURE
(LNM$PROCESS_DIRECTORY)
    (LNM$PROCESS_TABLE)
(LNM$SYSTEM_DIRECTORY)
    (LNM$GROUP_000360)
    (LNM$JOB_806E98E0)
    (LNM$SYSTEM_TABLE)
```

This example shows the process table, LNM$PROCESS_TABLE, in the process directory table, and the group, job, and system tables in the system directory table.

13.9 Deleting Logical Names and Logical Name Tables

13.9.1 Using the DEASSIGN Command

To delete a logical name, use the DEASSIGN command. When you define logical names in your process and job logical name tables, they are not deleted until your process terminates or they are explicitly deleted by user actions. However, if you specify the /USER_MODE qualifier to the DEFINE command, the logical name is defined in the process logical name table and deleted automatically after executing the next command image.

To delete a logical name ending with a colon, specify two colons. The DEASSIGN command, like the ASSIGN command, removes one colon before it searches the logical name table for a match.

To delete a logical name table, specify the table that contains it (the system or process directory logical name table) and the name of the table. All logical names in descendant tables (and the descendant tables themselves) are deleted when you delete a parent logical name table.

To delete a shareable logical name table, you must have DELETE access to the table or SYSPRV privilege.

13.9.2 Examples

- In the following example, the command deletes the logical name WORKFILE:

```
$ DEASSIGN WORKFILE
```

- In the following example, the logical name table TAX is deleted from the process directory table:

```
$ DEASSIGN/TABLE=LNM$PROCESS_DIRECTORY TAX
```

13.10 Redefining Process-Permanent Logical Names

13.10.1 Overview

Process-permanent logical names are created by DCL when you log in and they remain defined for the life of your process. You cannot deassign these logical names. You can redefine them (by specifying a different equivalence string in a DEFINE command) but if the redefined name is later deassigned, the process-permanent name is reestablished.

13.10.2 Available Process Permanent Logical Names

The following process-permanent logical names are available:

- SYS$INPUT

  Logical name that refers to the default input device or file
- SYS$OUTPUT

  Logical name that refers to the default output device or file
- SYS$ERROR

  Logical name that refers to the default device or file to which the system writes messages
- SYS$COMMAND

  Logical name that refers to the value of SYS$INPUT when you log in

13.10.3 Using the System Interactively

When you use the system interactively, DCL equates SYS$INPUT, SYS$OUTPUT, SYS$ERROR, and SYS$COMMAND to your terminal. However, when you execute command procedures and submit batch jobs, DCL creates new equivalence strings for these logical names.

13.10.4 Executing Command Procedures Interactively

When you execute a command procedure interactively, the following occur:

- SYS$INPUT is equated to the command procedure. Therefore, DCL obtains data from the command procedure. This assignment is temporary. After the command procedure terminates, SYS$INPUT regains its original value.
- SYS$OUTPUT, SYS$COMMAND, and SYS$ERROR remain equated to the terminal.

13.10.5 Submitting Batch Jobs

When you submit a batch job, the following occur:

- SYS$INPUT and SYS$COMMAND are equated to the batch job command procedure.
- SYS$OUTPUT and SYS$ERROR are equated to the batch job log file.

13.10.6 Nested Command Procedures

When you nest command procedures (that is, when you write a command procedure that executes other command procedures), the equivalence string for SYS$INPUT changes to point to the command procedure that is currently executing. However, the equivalence strings for SYS$OUTPUT, SYS$ERROR, and SYS$COMMAND remain the same unless you explicitly change them.

13.10.7 Opening Files

In addition, when you enter a command that opens a file, DCL opens the file as a process-permanent file. For example, if you open a file with the OPEN command, this file is opened as a process-permanent file. The file remains open until you explicitly close the file or until you log out.

Process-permanent files are stored in a special area in memory. Note that if you keep a large number of files open at the same time, you can exhaust this area. If this occurs, close some of the files (or log out).

13.11 Using Process-Permanent Logical Names as File Specifications

13.11.1 Overview

This section describes how to use process-permanent logical names as file specifications. In command procedures, you can use these names to read data from the terminal and to display data (see Chapter 15 and Chapter 16).

13.11.2 Redefining SYS$INPUT

You can redefine SYS$INPUT so that a command procedure reads input from the terminal or another file.

Note that if you redefine SYS$INPUT, DCL ignores your definition. DCL always obtains input from the default input stream. However, images such as command procedures can use your definition for SYS$INPUT.

13.11.3 Examples

In the following example, the commands shown are included in a command procedure to enable editing a file from a command procedure:

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ EDIT/TPU MYFILE.DAT
    .
    .
    .
```

SYS$INPUT is redefined as SYS$COMMAND so that the editor obtains input from the terminal rather than from the command procedure file (the default). SYS$COMMAND refers to the terminal, the initial input stream when you logged in. The /USER_MODE qualifier tells the command procedure that SYS$INPUT is redefined only for the duration of the next image. In this example, the next image is the editor. When the editor is finished, SYS$INPUT resumes its default value. In this case, the default value is the command procedure file.

13.11.4 Redefining SYS$OUTPUT

You can redefine SYS$OUTPUT to redirect output from your default device to another file. When you redefine SYS$OUTPUT, the system opens a file with the name you specify in the logical name assignment. When you define SYS$OUTPUT, all subsequent output is directed to the new file.

Remember to deassign SYS$OUTPUT or output will continue to be written to the file you specify. Note that you can redefine SYS$OUTPUT in user mode (with DEFINE/USER_MODE) to redirect output from an image. This definition is in effect only until the next command image is executed. Once the command image is executed (that is, the output is captured in a file), the logical name SYS$OUTPUT resumes its default value.

When you log in, the system creates two logical names called SYS$OUTPUT. One name is created in executive mode; the other name is created in supervisor mode. You can supersede the supervisor mode logical name by redefining SYS$OUTPUT. If you deassign the supervisor mode name, the system redefines SYS$OUTPUT in supervisor mode, using the executive mode equivalence string. You cannot deassign the executive mode name.

When you redefine SYS$OUTPUT to a file, the logical name contains only the device portion of the file specification even though the output is directed to the file you specify.

If the system cannot open the file you specify when you redefine SYS$OUTPUT, an error message displays.

After you redefine SYS$OUTPUT, most commands direct output to the existing version of the file. However, certain commands create a new version of the file before they write output.

13.11.5 Examples

- In the following example, SYS$OUTPUT is defined as MYFILE.LIS before the SHOW DEVICES command is entered. The display produced by SHOW DEVICES is directed to MYFILE.LIS in the current directory rather than to the terminal. The data can be manipulated as would any other text file:

```
$ DEFINE SYS$OUTPUT MYFILE.LIS
$ SHOW DEVICES
```

- In the following example, SYS$OUTPUT is redefined to the file TEMP.DAT. When SYS$OUTPUT is redefined, output from DCL and from images is directed to the file TEMP.DAT. The output from the SHOW LOGICAL command and from the SHOW TIME command is also sent to TEMP.DAT. When SYS$OUTPUT is deassigned, the system closes the file TEMP.DAT and redefines SYS$OUTPUT to the terminal. When the TYPE command is entered, the output collected in TEMP.DAT displays on the terminal:

```
$ DEFINE SYS$OUTPUT TEMP.DAT
$ SHOW LOGICAL SYS$OUTPUT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
$ TYPE TEMP.DAT
   "SYS$OUTPUT" = "DISK1:" (LNM$PROCESS_TABLE)
  06-MAY-1995 13:26:53
```

  When SYS$OUTPUT is redefined, the equivalence string contains the device name DISK1, not the full file specification.

13.11.6 Redefining SYS$ERROR

You can redefine SYS$ERROR to direct error messages to a specified file. However, if you redefine SYS$ERROR so it is different from SYS$OUTPUT (or if you redefine SYS$OUTPUT without also redefining SYS$ERROR), DCL commands send informational, warning, error, and severe error messages to both SYS$ERROR and SYS$OUTPUT. Therefore, you receive these messages twice—once in the file indicated by the definition of SYS$ERROR and once in the file indicated by SYS$OUTPUT. Success messages are sent only to the file indicated by SYS$OUTPUT.

DCL commands and images, which use standard error display mechanisms, send error messages to both SYS$ERROR and SYS$OUTPUT even when SYS$ERROR is different from SYS$OUTPUT. However, if you redefine SYS$ERROR, then run an image that references SYS$ERROR, the image sends error messages only to the file indicated by SYS$ERROR. This is true even if SYS$ERROR is different from SYS$OUTPUT.

13.11.7 Redefining SYS$COMMAND

Although you can redefine SYS$COMMAND, DCL ignores your definition. DCL always uses the default definition for your initial input stream. However, if you execute an image that references SYS$COMMAND, the image can use your new definition.

13.12 Logical Name Translation

13.12.1 Translation of Logical Names

When the system reads a file specification or device name in a DCL command line, it examines the file specification or device name to see whether the leftmost component is a logical name. If the leftmost component ends with a colon, space, comma, or a line terminator (for example, Return), the system attempts to

translate it as a logical name. If the leftmost component ends with any other character, the system does not attempt to translate it as a logical name.

13.12.2 Examples

- After you enter the command shown in this example, the system checks to see whether PUP is a logical name because PUP is the leftmost component of the file specification. Because the leftmost component is terminated with Return, the system attempts to translate PUP.

  ```
  $ TYPE PUP Return
  ```

- After you enter the command shown this example, the system checks whether DISK is a logical name. The system attempts to translate DISK because it is the leftmost component and ends with a colon. The system does not check PUP:

  ```
  $ TYPE DISK:PUP Return
  ```

- In the following example, the system does not try to translate [DRYSDALE]PUP because the leftmost component ends with a right square bracket (]):

  ```
  $ TYPE [DRYSDALE]PUP Return
  ```

13.12.3 Modifying Logical Name Translation

By default, when the system translates logical names in file specifications, it searches the process, job, group, and system tables in that order and uses the first match it finds.

There are two ways to change the default search order:

- Redefine the logical name LNM$FILE_DEV within the system directory table.
- Create a private definition of LNM$FILE_DEV within the process directory table.

13.12.4 Example

In the following example, the system searches the process and system tables only:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY -
_$ LNM$FILE_DEV LNM$PROCESS,LNM$SYSTEM
```

The equivalence strings for LNM$FILE_DEV do not include LNM$JOB and LNM$GROUP; therefore, subsequent commands that need to translate a logical name will not search the job or group tables.

13.12.5 Iterative Translation

Logical name translation can be iterative: after the system translates a logical name, it repeats the translation process for any logical names it finds contained within the first logical name.

The system limits the number of levels to which it performs logical name translation. The number of levels varies among system facilities but it is at least nine. If you define more than

the system-determined number of levels or if you create a circular definition, an error occurs when the logical name is used.

13.12.6 Example

In the following example, the first DEFINE command equates the logical name DISK to the device name DUA1. The second DEFINE command equates the logical name MEMO to the file specification DISK:[JEFF.MEMOS]COMPLAINT.TXT:

```
$ DEFINE DISK DUA1:
$ DEFINE MEMO DISK:[JEFF.MEMOS]COMPLAINT.TXT
```

When the system translates the logical name MEMO, it finds the equivalence string DISK:[JEFF.MEMOS]COMPLAINT.TXT. It then checks to see whether the leftmost component in this file specification ends in a colon, a space, a comma, or an end-of-line terminator. It finds a colon after DISK. The system translates that logical name also. The final translation of the file specification is:

```
DUA1:[JEFF.MEMOS]COMPLAINT.TXT
```

13.12.7 System Defaults During Logical Name Translation

When the system translates a logical name, it fills in any missing fields in a file specification with the current default device, directory, and version number. When you use a logical name to specify the input file for a command, the command uses the logical name to assign a file specification to the output file as well.

If the equivalence string contains a file name and file type, the output file is given the same file name and file type. If the equivalence string does not contain a file type, a default file type is supplied. The file type supplied depends on the command you are using.

When you use logical names in a list of input files, the equivalence string of each logical name provides a temporary default.

13.12.8 Example

In the following example, because a device name is not specified for the logical name HIG, the device name for MAL defines DBA1 as the temporary default device:

```
$ SET DEFAULT  DBA2:[CASEY]
$ DEFINE MAL DBA1:[MALCOLM]
$ DEFINE HIG [HIGGINS]
$ PRINT ALPHA,MAL:BETA,HIG:GAMMA
```

The PRINT command looks for the following files:

```
DBA2:[CASEY]ALPHA.LIS
DBA1:[MALCOLM]BETA.LIS
DBA1:[HIGGINS]GAMMA.LIS
```

13.13 Search Lists

13.13.1 Search List Translation

When you use a search list in a file specification, the search list is translated as follows:

- If the search list contains only a device, the original default directory is searched.
- If the search list contains a device and a directory, both are used to construct a complete file specification.

When you specify a search list as the first part of the parameter for the SET DEFAULT command, the system assigns the search list name, untranslated, to SYS$DISK. (SYS$DISK is a logical name that translates to your default disk.) Note that when you specify a search list as the first part of a parameter for the SET DEFAULT command, each equivalence string in the search list must contain a device name.

13.13.2 Examples

- In the following example, both a device and directory are specified; thus, they are both used to construct the file specifications:

```
$ DEFINE FIFI DISK1:[FRED],DISK2:[GLADYS],DISK3:[MEATBALL.SUB]
$ DIRECTORY FIFI:MEMO.LIS
```

  This command displays the following files:

  DISK1:[FRED]MEMO.LIS
  DISK2:[GLADYS]MEMO.LIS
  DISK3:[MEATBALL.SUB]MEMO.LIS

- At the beginning of this example, the default disk and directory are DISK2:[MEATBALL.SUB]. Next, a search list is defined. The SET DEFAULT command uses the search list as its parameter. When you enter the SHOW DEFAULT command a second time, the default directory has not changed. However, the search list FIFI is displayed as the default device along with its equivalence strings. The SHOW DEFAULT command displays the search list in the order in which the search list is evaluated by the system:

```
$ SHOW DEFAULT
  DISK2:[MEATBALL.SUB]
$ DEFINE FIFI DISK1:[FRED], DISK2:[GLADYS], DISK3:
$ SET DEFAULT FIFI
$ SHOW DEFAULT
  FIFI:[MEATBALL.SUB]
=   DISK1:[FRED]
=   DISK2:[GLADYS]
=   DISK3:[MEATBALL.SUB]
```

13.13.3 Search Lists with Wildcards

When you use a search list with a command that does not accept wildcards in a file specification, the system forms a file specification by using each equivalence string in the search list until a file specification for an existing file is found. The command affects only the first file found. The system displays an error message only after it checks all equivalence strings in a search list. Then, the system reports an error only on the last file it attempts to find.

13.13.4 Example

In the following example, the system forms the file specification DISK1:[FRED]QUOTAS.TXT and searches for that file:

```
$ DEFINE DECEMBER DISK1:[FRED],WORK2:[BARNEY]
$ EDIT/EDT DECEMBER:QUOTAS.TXT
```

If QUOTAS.TXT is found in DISK1:[FRED], it is opened for editing. No other files are subsequently opened. If QUOTAS.TXT is not found in DISK1:[FRED], the system searches for it in WORK2:[BARNEY]. If QUOTAS.TXT is found there, it is opened. If it is not found, an error message displays.

13.13.5 Using the RUN Command with Search Lists

When the RUN command is followed by a search list, the system forms file specifications as described previously. However, the system then checks to see whether any of the files in the list are installed images. The system runs the first file in the search list that is an installed image. Then, the RUN command terminates.

If none of the file specifications are installed images, the system repeats the process of forming file specifications. This time it looks for each file specification on the disk. It runs the first file it finds there. An error message is displayed if none of the specified files is found in either the known file list or on the disk.

13.13.6 Search Order for Multiple Search Lists

A file specification can contain more than one search list. When this occurs, each item in the file name search list is used, while the first device name is held constant. After all the items in the file name search list have been combined with the first device name, they are combined with the second device name. This continues until each device has been searched.

You can also have iterative (nested) search lists when one name in a search list translates to another search list. If this occurs, the system uses each name in a sublist before continuing to the next upper level name.

13.13.7 Examples

- In the following example, a file specification that has a search list in the file name and in the device name is shown:

```
$ DEFINE FILE CHAP1.RNO, CHAP2.RNO
$ DEFINE DISK WORK1:[ROSE], WORK2:[THORN]
$ SET DEFAULT DISK
$ DIRECTORY FILE

Directory WORK1:[ROSE]

CHAP1.RNO;2          CHAP2.RNO;1

Total of 2 files.

Directory WORK2:[THORN]

CHAP1.RNO;1          CHAP2.RNO;1

Total of 2 files.

Grand total of 2 directories, 4 files.
```

  The directory listing for each file name is given, first for WORK1:[ROSE] and second for WORK2:[THORN].

- In the following example, iterative search lists are shown:

```
$ DEFINE NESTED FRED.DAT, NEW_LIST, RICKY.DAT
$ DEFINE NEW_LIST ETHEL.DAT, LUCY.DAT
```

  The search order for the search list NESTED is as follows:

  FRED.DAT
  ETHEL.DAT
  LUCY.DAT
  RICKY.DAT

13.13.8 Logical Names in Multiple Tables

Multiple tables with the same name can exist. For example, there can be both a process-private and a shareable table called MY_TABLE. The process-private version always takes precedence over the shareable table in all logical name table processing. When a logical name such as LNM$FILE_DEV is used as a table name, the logical name is iteratively translated until a list of table names is formed. During this iterative translation, each name is first translated in the process directory. If this translation fails, it is then translated in the system directory. This order of precedence cannot be changed. As a consequence of this ordering, a logical name placed in the process directory table for use as a table name always takes precedence over any identical name residing in the system directory.

14

Symbols: Defining Commands and Expressions

14.1 Overview

A symbol is a name that represents a numeric, character, or logical value (such as true or false). When you use a symbol in a DCL command line, DCL replaces the symbol with its value before executing the command.

This chapter describes:

- Using symbols
- Displaying symbols
- Using symbols with other symbols
- Using symbols to store and manipulate data
- Character strings
- Using numeric values and expressions
- Using logical values and expressions
- Converting value types in expressions
- Understanding symbol tables
- Masking the value of symbols
- Understanding symbol substitution
- The three phases of command processing
- An alternative to using symbols: automatic foreign commands

14.1.1 References

- Additional information on symbols and their usage can be found in the *OpenVMS DCL Dictionary*.
- Information on defining new commands can be found in the *OpenVMS Command Definition, Librarian, and Message Utilities Manual*.

14.2 About Symbols

14.2.1 Symbol Usage

You can use symbols in the following ways:

- As synonyms for commands, parameters, or command lines. Instead of typing a long command line, you can create a symbol to use instead.
- To define a foreign command, which allows you to execute an image by entering only the symbol name. The command is ''foreign'' because it is unknown to DCL.
- In command procedure files, to perform programming tasks such as conditional execution and substitution of variables.

  You can use symbols as variables in expressions or to pass parameters to and from command procedures. In addition, DCL commands such as READ, WRITE, and INQUIRE use symbols to refer to data records.

14.2.2 Example

In the following example, a symbol is created to set default to a directory that is accessed often. These commands show how to define and use the symbol WORK to set default to the WORK1:[JONES.WORK] directory:

```
$  WORK :== SET DEFAULT DISK1:[JONES.WORK]
$  WORK
$  SHOW DEFAULT
   DISK1:[JONES.WORK]
```

14.2.3 Comparing Logical Names and Symbols

Although logical names and symbols appear similar, they are used differently. The following table compares the function, usage, and other characteristics of logical names and symbols:

Characteristic	Logical Names	Symbols
Function	Represent device, directory, file, queue, and other system object specifications.	Represent commands or portions of command strings.
Usage	Are used in place of any complete device, directory, file, queue or other system object specification. Logical names must be used as part of a command string parameter to be passed to the file system for translation.	Are used in place of any command string. Symbols must be used as the first word in a command string to be translated by the command language interpreter.
Storage	Are stored in your process, job, group, or system logical name table. See Section 13.5.	Are stored in your global or local symbol table. See Section 14.11.

Characteristic	Logical Names	Symbols
Creation	Use either the ASSIGN or DEFINE command to create a logical name. See Section 13.4.	Use an assignment statement (= or ==) to create a symbol. See Section 14.3.
Display	Use either the SHOW LOGICAL or SHOW TRANSLATION command to display a logical name. See Section 13.8.	Use the SHOW SYMBOL command to display a symbol. See Section 14.4.
Deletion	Use the DEASSIGN command to delete a logical name. See Section 13.9.	Use the DELETE/SYMBOL command to delete a symbol. See Section 14.3.12.

14.3 Using Symbols

14.3.1 Types of Symbols

You can create two types of symbols, local and global. **Local symbols** are accessible from the current command level and from command procedures executed from the current command level. **Global symbols** are accessible at all command levels.

14.3.2 Defining Symbols

You can define a symbol with a character string, a number, a lexical function, a logical value, or another symbol. The symbol name can be 1 to 255 characters long and must begin with a letter, an underscore (_), or a dollar sign ($). In a symbol name, both lowercase and uppercase letters are treated as uppercase.

14.3.3 Using the Assignment Statement

To create a symbol, use the **assignment statement** (= or ==) or the string assignment (:= or :==). When you use the string assignment, all alphabetic characters are converted to uppercase and multiple spaces and tabs are compressed to a single space. You can use string assignments to create a symbol that represents a DCL command or to define a foreign command (note that in either case, there is a 255-character limit). To continue a character string over two lines in a string assignment, use a single hyphen.

You can also create symbols by using the READ and INQUIRE commands (see Chapter 15 and Chapter 16).

14.3.4 Examples: Creating Local Symbols

- In the following example, the local symbol SS is assigned to the DCL command SHOW SYMBOL:

```
$ SS = "SHOW SYMBOL"
```

- In the following example, the local symbol DB is assigned to the DCL command DIRECTORY ACCOUNTS:[BOLIVAR]:

```
$ DB := DIRECTORY ACCOUNTS:[BOLIVAR]
```

14.3.5 Examples: Creating Global Symbols

- In the following example, the global symbol DC is used to represent a DCL command line. The DCL command DIRECTORY is executed with the specified qualifiers when you enter the symbol name:

```
$ DC == "DIRECTORY/SIZE=ALL DISK1:[JONES.TAX]MONEY.LIS"
```

- In the following example, the global symbol READY is used to represent a DCL command line. The DCL command PRINT is executed with the specified qualifiers when you enter the symbol name:

```
$ READY :== PRINT/CONFIRM/QUEUE=AKI$LN03/NOTIFY/RESTART
$ READY FILE.DAT
```

14.3.6 Using Symbols to Represent DCL Commands

You can define a symbol to represent a DCL command in your login command file (LOGIN.COM) or interactively at DCL level. When you define the symbol in your login command file, you can use the symbol each time you log in; when you define the symbol interactively, the symbol can be used only during the current process.

If you define a symbol with the same name as a DCL command, your definition overrides the DCL command name. For example, if you define the symbol HELP as the command TYPE HELP.LST, you can no longer invoke the system's Help utility by typing HELP.

14.3.7 Symbol Abbreviation

Use the asterisk (*) to create a symbol that can be abbreviated. Generally, you can use abbreviated symbol definitions in any situation that allows a symbol to be used. Symbols that involve substring replacement are an exception. See Section 14.7.13 for more information.

Note that existing symbols might be superseded. If an existing symbol exactly matches the new symbol at or past the asterisk, the new symbol replaces the existing symbol. In addition, you cannot define another symbol whose name partly matches the existing symbol at or past the asterisk.

14.3.8 Example

The following example creates the local symbol PRINT, which can be abbreviated as PR, PRI, or PRIN:

```
$ PR*INT = "PRINT/CONFIRM/QUEUE=AKI$LN03/NOTIFY/RESTART"
```

To execute the DCL command PRINT with the specified qualifiers, you can enter the symbol or any of its abbreviations.

14.3.9 Defining Foreign Commands

If you equate the file specification of a non-DCL image to a symbol, you can run the image by typing the symbol name. A symbol that runs an image is referred to as a foreign command. A foreign command is an image that is not recognized by the command interpreter as a DCL command. (Note that, like each element of a DCL command, a foreign command has a 255-character limit.)

The formats for defining a symbol as a foreign command are as follows:

```
symbol-name :=[=] $image-file-spec
symbol-name =[=] "$image-file-spec"
```

Note that when the dollar sign ($) precedes a file specification at the beginning of a symbol definition, without any space between the dollar sign and the file specification, the request to run the image is implied.

For the image file specification, the default device and directory name is SYS$SYSTEM, the default file type is .EXE, and the default file version number is the highest version.

An alternative to using a foreign command is to define new commands with the Command Definition utility. See the *OpenVMS Command Definition, Librarian, and Message Utilities Manual* for more information.

There is also a method for executing foreign commands automatically, without specifying symbols. See Section 14.15 for more information.

14.3.10 Example

- In the following example, the global symbol PRINTALL is defined to execute the image DISK1:[ACCOUNTS]PRINTALL.EXE:

  ```
  $ PRINTALL :==   $[ACCOUNTS]PRINTALL
  ```

  In a command line, PRINTALL could be followed by a parameter.

- In the following example, the file specification RAT.DAT is a parameter that is passed to the image defined by PRINTALL:

  ```
  $ PRINTALL RAT.DAT
  ```

14.3.11 Symbol Substitution

The command interpreter looks for symbols enclosed by apostrophes (') and translates them. Thus, if you use symbols or lexical functions preceded by apostrophes to specify parameters, symbol substitution occurs (see Section 14.13). Otherwise, the command interpreter does not parse the line. The image must obtain the parameter and perform any **parsing** or evaluation of the command line.

14.3.12 Deleting Symbols

The DELETE/SYMBOL command deletes a symbol. To delete a global symbol, include the /GLOBAL qualifier. For example, to delete the global symbol TEMP, enter the following command:

```
$ DELETE/SYMBOL/GLOBAL TEMP
```

14.4 Displaying Symbols

14.4.1 Using the SHOW SYMBOL Command

The SHOW SYMBOL command displays the values of symbols. To display the value of a particular symbol, enter the SHOW SYMBOL command followed by the name of the symbol. To display the value of a particular global symbol, include the /GLOBAL qualifier. The SHOW SYMBOL/ALL command displays all local symbols. The command SHOW SYMBOL/ALL/GLOBAL displays all global symbols.

Note that when a symbol has an integer value, the SHOW SYMBOL command displays the value in decimal, hexadecimal, and octal notation.

14.4.2 Examples

- In the following example, the symbol PR is displayed:

```
$  SHOW SYMBOL PR
PR*INT = "PRINT/CONFIRM/COPIES=2/QUEUE=DOC$LN03/NOTIFY/RESTART"
```

- In the following example, the integer value for the symbol TOTAL is displayed:

```
$ SHOW SYMBOL TOTAL
  TOTAL = 4   Hex = 00000004  Octal = 00000000004
```

14.5 Using Symbols with Other Symbols

14.5.1 Defining a Symbol as a Symbol

After you define a symbol, you can use it as part of the definition of another symbol. DCL interprets a symbol as a character string or a number, depending on the context in which you use the symbol.

14.5.2 Example

In the following example, the integer value 3 is assigned to the symbol COUNT:

```
$ COUNT = 3
```

The value of COUNT can then be used in other assignment statements. For example, here the value of COUNT is added to 1:

```
$ TOTAL = COUNT + 1
```

The result (4) is equated to the symbol TOTAL.

14.5.3 Symbol Concatenation

You can concatenate several symbols to create a long character string by using the plus sign (+). You can also concatenate two or more symbols by placing apostrophes (') around each symbol name.

For more information on requesting symbol substitution, see Section 14.13.8.

14.5.4 Examples

- In the following example, the symbols "Saturday," and "Sunday" are used to create the symbol "WEEKEND":

```
$ DAY1 = "Saturday, "
$ DAY2 = "Sunday"
$ WEEKEND = DAY1 + DAY2
$ SHOW SYMBOL WEEKEND
   WEEKEND = "Saturday, Sunday"
```

- In the following example, apostrophes are used to concatenate the symbols NAME and TYPE:

```
$ NAME = "MYFILE"
$ TYPE = ".DAT"
$ PRINT 'NAME''TYPE'
```

  The PRINT command prints a copy of MYFILE.DAT.

14.5.5 Including Symbols in String Assignments

To include a local symbol in a string assignment, use a colon and an equal sign (:=). To include a global symbol in a string assignment, use a colon and two equal signs (:==). For either type of symbol (local or global), enclose the symbol in apostrophes (' '). Otherwise, DCL will not recognize it as a symbol.

If you define a null character string for a symbol, that symbol has a value of 0.

14.5.6 Examples

- In the following example, the symbol COUNT is included in a string assignment statement:

```
$ BARK := P'COUNT'
```

  In a previous example, COUNT was assigned the integer value 3. In this example, COUNT is converted to a string value and appended to the character P. The local symbol BARK now has the value P3.

- In the following example, the symbol A is null:

```
$ A = ""
$ B = 2
$ C = A + B
$ SHOW SYMBOL C
   C = 2  Hex = 00000002  Octal = 00000000002
```

14.6 Using Symbols to Store and Manipulate Data

14.6.1 Variables

You can use symbols as variables in command procedures. Variables hold values that you calculate or assign as something other than a literal value. For example, you might assign the value of a lexical function to a variable or read the value of a file record into a variable.

14.6.2 Expressions

An **expression** is a combination of values. In command procedures, expressions are used in symbol assignment statements (on the right side of the equal sign), in IF statements, in WRITE commands, and as arguments for lexical functions.

When you define a symbol, the left side of the assignment statement defines the symbol name; the right side of the assignment statement contains an expression. Each value (also called an **operand**) in an expression can be connected to another value by an **operator**. DCL evaluates the expression and assigns the result to the symbol. If an expression is evaluated as a character string, then the symbol has a string value.

14.6.3 Example

In the following example, the local symbol BARK is equated to an expression that adds three numbers:

```
$ BARK = 1 + 2 + 3
```

The operands are 1, 2, and 3. The operator is the plus sign (+). The evaluated expression is an integer, so the symbol has an integer value.

14.7 Character Strings

14.7.1 Overview

A character string can contain any characters that can be printed. Appendix B includes tables of the ASCII character set and the DEC Multinational character set. These tables list characters you can include in a character string.

14.7.2 Types of Characters

Characters fall into three main categories:

- Alphanumeric characters

  The uppercase letters A to Z, lowercase letters a to z, digits 1 to 9, dollar sign ($), underscore (_), and hyphen (-).

- Special characters

  All other characters that can be displayed or printed: exclamation point (!), quotation marks ("), number sign (#), and so on.

- Nonprintable characters

All characters that cannot be printed or displayed. In general, nonprintable characters are ignored for display and print purposes. However, several nonprintable characters serve control functions as follows:

Character	Function
HT	Starts printing or typing at the next horizontal tab
LF	Starts printing or typing on the next line
FF	Starts printing or typing at the top of the next page
CR	Starts printing or typing at the first space on the same line
ESC	Introduces a terminal escape sequence
SP	Inserts one space

14.7.3 Defining Character Strings

You can define a character string by enclosing it in quotation marks (" "). In this way, alphabetic case and spaces are preserved when the symbol assignment is made. Note the following:

- To include quotation marks (") within a string, type two consecutive quotation marks.
- To continue a character string over two lines, use a plus sign (for string concatenation) and a hyphen (for continuation).

  You cannot use the hyphen continuation character within a quoted character string.

14.7.4 Examples

- In the following example, the string "YES" is quoted, so it must be defined within quotation marks:

```
$ PROMPT = "Type ""YES"" or ""NO"""
$ SHOW SYMBOL PROMPT
  PROMPT = "Type "YES" or "NO""
```

- In the following example, the character string is continued over two lines:

```
$ HEAD = "MONTHLY REPORT FOR" + -
_$ " DECEMBER 1995"
$ SHOW SYMBOL HEAD
  HEAD = "MONTHLY REPORT FOR DECEMBER 1995"
```

14.7.5 Character String Expressions

A character string expression can contain character strings, lexical functions that are evaluated as character strings, or symbols that have character string values. When you use a character string in an expression, you must enclose it in quotation marks (" "). If you do not use quotation marks, DCL processes the string as a symbol.

14.7.6 Character String Expression Operands

Character string expressions combine the following values (called string operands):

- Character strings that must appear in quotation marks
- Symbols that represent character strings
- Lexical functions that are evaluated as character strings

If you perform an operation or comparison between a character string and a number, DCL converts the character string to a number.

String operands can be added (string concatenation), subtracted (string reduction), compared, or replaced with other character strings as described in the following subsections.

14.7.7 Examples

- In the following example, the character string "CAT" must appear in quotation marks:

```
$  TEMP = "CAT"
```

- In the following example, the symbol TEMP represents the character string "CAT". The symbol TOPIC is a concatenation of the character string "THE" and the character string that the symbol TEMP represents ("CAT"). The result is "THE CAT".

```
$  TOPIC = "THE" + TEMP
```

- In the following example, the symbol COUNT represents the lexical function F$STRING(65):

```
$  COUNT = F$STRING(65)
```

14.7.8 Character String Operations

You can specify the following character string operations:

- Concatenation—The plus sign concatenates two character strings.
- Reduction—The minus sign removes the second character string from the first character string.

  If the second character string occurs more than once in the first character string, only the first occurrence of the string is removed.

14.7.9 Examples

- In the following example, the plus sign (+) is used to concatenate two character strings:

```
$ COLOR = "light brown"
$ WEIGHT = "30 lbs."
$ DOG2 = "No tag, " + COLOR + ", " + WEIGHT
$ SHOW SYMBOL DOG2
  DOG2 = "No tag, light brown, 30 lbs."
```

- In the following example, the minus sign (–) is used to remove a character string:

```
$ SHOW SYMBOL DOG2
  DOG2 = "No tag, light brown, 30 lbs."
$ DOG2 = DOG2 - ", 30 lbs."
$ SHOW SYMBOL DOG2
  DOG2 = "No tag, light brown"
```

14.7.10 Comparing Character Strings

When you compare two character strings, the strings are compared character by character. Strings of different lengths are not equal (for example, ‘‘dogs’’ is greater than ‘‘dog’’).

The comparison criteria are the ASCII values of the characters. Under these criteria, the digits 0 to 9 are less than the uppercase letters A to Z, and the uppercase letters A to Z are less than the lowercase letters a to z. A character string comparison ends when either of the following conditions is true:

- All the characters have been compared, in which case the strings are equal.
- The first mismatch occurs.

14.7.11 Types of String Comparisons

Table 14–1 lists different types of string comparisons.

Table 14–1 String Comparisons

Comparison	Operator	Description
Equal to	.EQS.	Compares one character string to another for equality.
Greater than or equal to	.GES.	Compares one character string to another for greater or equal value in the first specified string.
Greater than	.GTS.	Compares one character string to another for a greater value in the first specified string.
Less than or equal to	.LES.	Compares one character string to another for a lesser or equal value in the first specified string.
Less than	.LTS.	Compares one character string to another for a lesser value in the first specified string.
Not equal	.NES.	Compares one character string to another for inequality.

14.7.12 Examples

In all of these examples, assume that the symbol LAST_NAME has the value ‘‘WHITFIELD’’.

- In the following example, the symbol TEST_NAME is evaluated as 0 (False); the value of the symbol LAST_NAME does not equal the literal ‘‘HILL’’:

```
$ TEST_NAME = LAST_NAME .EQS. "Hill"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 0    . . .
```

- In the following example, the symbol TEST_NAME is evaluated as 1 (True); the value of the symbol LAST_NAME is greater than or equal to the literal ‘‘HILL’’:

```
$ TEST_NAME = LAST_NAME .GES. "HILL"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 1    . . .
```

- In the following example, the symbol TEST_NAME is evaluated as 1 (True); the value of the symbol LAST_NAME is greater than the literal ‘‘HILL.’’:

```
$ TEST_NAME = LAST_NAME .GTS. "HILL"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 1    . . .
```

- In the following example, the symbol TEST_NAME is evaluated as 0 (False); the value of the symbol LAST_NAME is not less than or equal to the literal ‘‘HILL’’:

```
$ TEST_NAME = LAST_NAME .LES. "HILL"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 0    . . .
```

- In the following example, the symbol TEST_NAME is evaluated as 0 (False); the value of the symbol LAST_NAME is not less than the literal ‘‘HILL’’:

```
$ TEST_NAME = LAST_NAME .LTS. "HILL"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 0    . . .
```

- In the following example, the symbol TEST_NAME is evaluated as 1 (True); the value of the symbol LAST_NAME does not equal the literal ‘‘HILL’’:

```
$ TEST_NAME = LAST_NAME .NES. "HILL"
$ SHOW SYMBOL TEST_NAME
  TEST_NAME = 1    . . .
```

14.7.13 Replacing Substrings

You can replace part of a character string with another character string by specifying the position and size of the replacement string. The format for local symbols is:

symbol-name[offset,size] := replacement-string

The format for global symbols is:

symbol-name[offset,size] := = replacement-string

The elements are as follows:

offset An integer that indicates the position of the replacement string relative to the first character in the original string. An offset of 0 means the first character in the symbol, an offset of 1 means the second character, and so on.

size An integer that indicates the length of the replacement string.

14.7.14 Rules for Replacing Substrings

To replace substrings, observe the following rules:

- The square brackets are required notation. No spaces are allowed between the symbol name and the left bracket.
- Integer values for size and offset values can range from 0 to 768.
- The replacement string must be a character string.
- The symbol name you specify can be undefined initially. The assignment statement creates the symbol name and if necessary, provides leading or trailing spaces in the symbol value.
- You can specify an offset and size to create a symbol that represents a blank line.

Lining up records in columns makes a list easier to read and sort. You can use this format to specify how you want data to be stored.

14.7.15 Examples

- In the following example, the first assignment statement gives the symbol A the value PACKRAT. The second statement specifies that MUSK replace the first four characters in the value of A. The result is that the value of A becomes MUSKRAT.

```
$ A := PACKRAT
$ A[0,4] := MUSK
$ SHOW SYMBOL A
   A = "MUSKRAT"
```

- In the following example, the symbol B does not have a previous value, so it is given a value of four leading spaces followed by RAT:

```
$ B[4,3] := RAT
```

- In the following example, a value of 80 blank spaces is assigned to the symbol LINE:

```
$ LINE[0,80]:= " "
```

- In the following example, the first statement fills in the first 15 columns of DATA with whatever value NAME has. The second statement fills in column 18 with whatever value GRADE has. Columns 16 and 17 contain blanks:

```
$ DATA[0,15] := 'NAME'
$ DATA[17,1] := 'GRADE'
```

14.8 Using Numeric Values and Expressions

14.8.1 Overview

A number can have the following values:

- Decimal—the ASCII characters 0 to 9
- Hexadecimal—the ASCII characters 0 to 9 and A to F
- Octal—the ASCII characters 0 to 7

The number you assign to a symbol must be in the range –2147483648 to 2147483647 (decimal). An error is not reported if a number outside this range is specified or calculated but an incorrect value results.

14.8.2 Specifying Numbers

At DCL command level and within command procedures, specify a number as follows:

- Positive numbers

  Specify a positive number by typing the appropriate digits.

- Negative numbers

  Precede a negative number with a minus sign (–).

- Radix

  Specify a number in a radix other than decimal by preceding the number (but not the minus sign) with %X for hexadecimal numbers and %0 for octal numbers.

- Fractions

  A number cannot include a decimal point. In calculations, fractions are truncated. For example, 8 divided by 3 equals 2.

14.8.3 Examples

- In the following example, the number 13 is assigned to the symbol DOG_COUNT:

```
$ DOG_COUNT = 13
$ SHOW SYMBOL DOG_COUNT
  DOG_COUNT = 13   Hex = 0000000D  Octal = 00000000015
```

- In the following example, the negative number (−15237) is represented with a minus sign (–):

```
$ BALANCE = -15237
$ SHOW SYMBOL BALANCE
  BALANCE = -15237   Hex = FFFFC47B  Octal = 37777742173
```

- In the following example, the hexadecimal number D is represented with the prefix %X:

```
$ DOG_COUNT = %XD
$ SHOW SYMBOL DOG_COUNT
  DOG_COUNT = 13   Hex = 0000000D  Octal = 00000000015
$ BALANCE = -%X3B85
$ SHOW SYMBOL BALANCE
  BALANCE = -15237   Hex = FFFFC47B  Octal = 37777742173
```

14.8.4 Internal Storage of Numbers

Numbers are stored internally as signed 4-byte integers, called longwords; positive numbers have values of 0 to 2147483647 and negative numbers have values of 4294967296 minus the absolute value of the number. The number −15237, for example, is stored as 4294952059. Negative numbers are converted back to minus-sign format for ASCII or decimal displays; however, they are not converted back for hexadecimal and octal displays. For example, the number −15237 appears in displays as hexadecimal FFFFC47B (decimal 4294952059) rather than hexadecimal –00003B85.

Numbers are stored in text files as a series of digits using ASCII conventions (for example, the digit 1 has a storage value of 49).

14.8.5 Numeric Expressions

In a **numeric expression**, the values involved must be literal numbers (such as 3) or symbols with numeric values. In addition, you can use a character string that represents a number (for example, "23" or "−51"). If you perform an operation or comparison between a number and a character string, DCL converts the character string to a number.

14.8.6 Integer Operands

Numeric expressions combine the following values (called integer operands):

- Integers. For example:

```
$  COUNT = 1
```

- Lexical functions that are evaluated as integers. For example:

```
$  B = F$INTEGER("-9" + 23)
```

- Symbols that have integer values. For example:

```
$  A = B - 6
```

  In the preceding example, the symbol B represents the integer value returned by the F$INTEGER function (− 923).

These integer operands can be connected by arithmetic, logical, and comparison operators, as described in the following sections.

14.8.7 Performing Arithmetic Operations

You can specify the following arithmetic operations:

- Multiplication

  The asterisk (*) multiplies two numbers.

- Division

  The slash (/) divides the first specified number by the second specified number. If a number does not divide evenly, the remainder is lost. No rounding takes place.

- Addition

  The plus sign (+) adds two numbers.

- Subtraction

  The minus sign (-) subtracts the second specified number from the first specified number.

- Unary plus and minus

  The plus and minus signs change the sign of the number they precede.

14.8.8 Examples

- The following example demonstrates using multiplication when assigning a symbol:

```
$ BALANCE = 142 * 14
$ SHOW SYMBOL BALANCE
  BALANCE = 1988   Hex = 000007C4  Octal = 00000003704
```

- The following example demonstrates using division when assigning a symbol:

```
$ BALANCE = BALANCE / 14
$ SHOW SYMBOL BALANCE
  BALANCE = 142   Hex = 0000008E  Octal = 00000000216
```

- The following example demonstrates using addition when assigning a symbol:

```
$ BALANCE = BALANCE + 37
$ SHOW SYMBOL BALANCE
  BALANCE = 179   Hex = 000000B3  Octal = 00000000263
```

- The following example demonstrates using subtraction when assigning a symbol:

```
$ BALANCE = BALANCE - 15416
$ SHOW SYMBOL BALANCE
  BALANCE = -15237   Hex = FFFFC47B  Octal = 00000142173
```

- The following example demonstrates using a unary minus sign to change the sign of the number -142:

```
$ BALANCE = -(-142)
$ SHOW SYMBOL BALANCE
  BALANCE = 142   Hex = 0000008E   Octal = 00000000216
```

14.8.9 Comparing Numbers

Table 14–2 lists different types of numeric comparisons:

Table 14–2 Numeric Comparisons

Comparison	Operator	Description
Equal to	.EQ.	Compares one number to another for equality.
Greater than or equal to	.GE.	Compares one number to another for a greater or equal value in the first number.
Greater than	.GT.	Compares one number to another for a greater value in the first number.
Less than or equal to	.LE.	Compares one number to another for a lesser or equal value in the first number.
Less than	.LT.	Compares one number to another for a lesser value in the first number.
Not equal to	.NE.	Compares one number to another for inequality.

14.8.10 Examples

In the following examples, assume that the symbol BALANCE has the value −15237.

- In the following example, TEST_BALANCE is evaluated as 1 (True); BALANCE equals −15237:

```
$ TEST_BALANCE = BALANCE .EQ. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 1    . . .
```

- In the following example, TEST_BALANCE is evaluated as 1 (True); BALANCE is greater than or equal to −15237:

```
$ TEST_BALANCE = BALANCE .GE. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 1    . . .
```

- In the following example, TEST_BALANCE is evaluated as 0 (False); BALANCE is not greater than −15237:

```
$ TEST_BALANCE = BALANCE .GT. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 0   . . .
```

- In the following example, TEST_BALANCE is evaluated as 1 (True); BALANCE is less than or equal to −15237:

```
$ TEST_BALANCE = BALANCE .LE. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 1    . . .
```

- In the following example, TEST_BALANCE is evaluated as 0 (False); BALANCE is not less than −15237:

```
$ TEST_BALANCE = BALANCE .LT. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 0    . . .
```

- In the following example, TEST_BALANCE is evaluated as 0 (False); BALANCE equals −15237:

```
$ TEST_BALANCE = BALANCE .NE. -15237
$ SHOW SYMBOL TEST_BALANCE
  TEST_BALANCE = 0    . . .
```

14.8.11 Performing Numeric Overlays

You can perform binary (bit-level) overlays of the current symbol value by using a special format of the assignment statement. For local symbols, the format is:

symbol-name[bit-position,size] = replacement-expression

For global symbols, the format is:

symbol-name[bit-position,size] == replacement-expression

The elements are as follows:

bit-position	An integer that indicates the location relative to bit 0 at which the overlay is to occur.
size	An integer that indicates the number of bits to be overlaid.

14.8.12 Rules for Using Numeric Overlays

To use numeric overlays, observe the following rules:

- The square brackets ([]) are required notation. No spaces are allowed between the symbol name and the left bracket.
- Literal values are assumed to be decimal.
- The maximum length for size is 32 bits.
- Replacement expression must be a numeric expression.
- When symbol-name is either undefined or defined as a string, the result of the overlay is a string. Otherwise, the result is an integer.

14.8.13 Example

The following example defines the symbol BELL as the value 7. The low-order byte of BELL has the binary value 00000111. By changing the 0 at offset 5 to 1 (beginning with 0, count bits from right to left), you create the binary value 00100111 (decimal value 39):

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
 BELL = 39   Hex = 00000027   Octal = 00000000047
```

14.9 Using Logical Values and Expressions

14.9.1 Logical Operations

Some operations interpret numbers and character strings as logical data with values as follows:

- True

A number has a logical value of true if it is odd (that is, the low-order bit is 1). A character string has a logical value of true if the first character is an uppercase or lowercase T or Y.

- False

  A number has a logical value of false if it is even (that is, the low-order bit is 0). A character string has a logical value of false if the first character is not an uppercase or lowercase T or Y.

14.9.2 Example

In the following examples, DOG_COUNT is assigned the value 13. IF STATUS means if the logical value of STATUS is true.

```
$ STATUS = 1
$ IF STATUS THEN DOG_COUNT = 13

$ STATUS = "TRUE"
$ IF STATUS THEN DOG_COUNT = 13
```

14.9.3 Logical Expressions

A logical operation affects all the bits in the number being acted upon. The values for **logical expressions** are integers, and the result of the expression is an integer as well. If you specify a character string value in a logical expression, the string is converted to an integer before the expression is evaluated.

14.9.4 Uses of Logical Expressions

Typically, you use logical expressions to evaluate the low-order bit of a logical value; that is, to determine whether the value is true or false. You can specify the following logical operations:

- .NOT.

  The operator .NOT. reverses the bit configuration of a logical value. A true value becomes false and a false value becomes true.

- .AND.

  The operator .AND. combines two logical values as follows:

Bit Level	Entity Level
1 .AND. 1 = 1	true .AND. true = true
1 .AND. 0 = 0	true .AND. false = false
0 .AND. 1 = 0	false .AND. true = false
0 .AND. 0 = 0	false .AND. false = false

- .OR.

  The operator .OR. combines two logical values as follows:

Bit Level	Entity Level
1 .OR. 1 = 1	true .OR. true = true
1 .OR. 0 = 1	true .OR. false = true
0 .OR. 1 = 1	false .OR. true = true
0 .OR. 0 = 0	false .OR. false = false

14.9.5 Examples

- The following example reverses a true value to false. The expression is evaluated as –2; the value is even and is therefore false:

```
$ SHOW SYMBOL STATUS
  STATUS = 1   Hex = 00000001  Octal = 00000000001
$ STATUS = .NOT. STATUS
$ SHOW SYMBOL STATUS
  STATUS = -2   Hex = FFFFFFFE  Octal = 37777777776
```

- The following example combines a true value and a false value to produce a false value:

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .AND. STAT2
$ SHOW SYMBOL STATUS
  STATUS = 0   Hex = 00000000  Octal = 00000000000
```

- The following example combines a true value and a false value to produce a true value:

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .OR. STAT2
$ SHOW SYMBOL STATUS
  STATUS = 1   Hex = 00000001  Octal = 00000000001
```

14.9.6 Logical Operation Results

The following tables demonstrate the results of logical operations on a bit-by-bit basis and a number-by-number basis. In logical operations, a character string beginning with an uppercase or lowercase T or Y is treated as the number 1; a character string beginning with any other character is treated as the number 0. In logical operations, odd numbers are true and even numbers and zero are false.

Given That:		The Results Are:		
Bit A	Bit B	.NOT. A	A .AND. B	A .OR. B
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Given That:		The Results Are:		
Number A	Number B	.NOT. A	A .AND. B	A .OR. B
odd	odd	even	odd	odd
odd	even	even	even	odd
even	odd	odd	even	odd
even	even	odd	even	even

14.9.7 Using Values Returned by Lexical Functions

Typically used in command procedures, lexical functions retrieve information from the system, including information about system processes, batch and print queues, and user processes. You can also use lexical functions to manipulate character strings and translate logical names. When you assign a lexical function to a symbol, the symbol is equated to the information returned by the lexical function (for example, a number or character string). At DCL level, you can then display that information with the DCL command SHOW SYMBOL. In a command procedure, the information stored in the symbol can be used later in the procedure. See the Chapter 17 for additional information on lexical functions.

14.9.8 Lexical Function Format

To use a lexical function, specify the name of the lexical function (which always begins with F$) and its argument list. Use the following format:

```
F$function-name(args[,...])
```

The argument list follows the function name with any number of intervening spaces and tabs.

14.9.9 Rules for Using Lexical Functions

When you use a lexical function, observe the following rules:

- Enclose the argument list in parentheses.
- Within the list, specify arguments in exact order and separate them with commas; even if you omit an optional argument, do not omit the comma.
- If no arguments are required, type an empty set of parentheses.
- Follow the rules for writing expressions: enclose character strings in quotation marks; do not enclose integers, symbols, and lexical functions in quotation marks.

14.9.10 Evaluation of Lexical Functions

Use lexical functions the same way you would use character strings, integers, and symbols. When you use a lexical function in an expression, DCL automatically evaluates the function and replaces the function with its return value.

14.9.11 Example

In the following example, the F$LENGTH function returns the length of the value specified as BUMBLEBEE as its argument. DCL automatically determines the return value (9) and uses this value to evaluate the expression. Therefore, the result of the expression (9 + 1) is 10 and this value is assigned to the symbol SUM:

```
$ SUM = F$LENGTH("BUMBLEBEE") + 1
$ SHOW SYMBOL SUM
  SUM = 10   Hex = 0000000A  Octal = 00000000012
```

14.9.12 Lexical Function Usage

Note that each lexical function returns information as either an integer or a character string. In addition, you must specify the arguments for a lexical function as either integer or character string expressions.

For example, the F$LENGTH function requires an argument that is a character string expression and it returns a value that is an integer. In a previous example, the argument "BUMBLEBEE" is a character string expression and the return value (9) is an integer.

You can use a lexical function in any position that you can use a symbol. In positions where symbol substitution must be forced by enclosing the symbol in apostrophes (see Section 14.13), lexical function evaluation must be forced by placing the lexical function within apostrophes. Lexical functions can also be used as argument values in other lexical functions.

14.9.13 Examples

The following examples show different ways you can specify the argument for the F$LENGTH function. In each example, the argument is a character string expression.

- The following example shows a symbol that is used as an argument:

  ```
  $ BUG = "BUMBLEBEE"
  $ LEN = F$LENGTH(BUG)
  $ SHOW SYMBOL LEN
    LEN = 9   Hex = 00000009  Octal = 00000000011
  ```

  When you use the symbol BUG as an argument, do not place quotation marks around it. The lexical function automatically substitutes the value "BUMBLEBEE" for BUG, determines the length, and returns the value 9.

- The following example shows an argument that contains both a symbol and a character string:

  ```
  $ BUG = "BUMBLEBEE"
  $ LEN = F$LENGTH(BUG)
  $ SHOW SYMBOL LEN
    LEN = 9   Hex = 00000009  Octal = 00000000011
  $ LEN = F$LENGTH(BUG + "S")
  $ SHOW SYMBOL LEN
    LEN = 10   Hex = 0000000A  Octal = 00000000012
  ```

The symbol BUG is not enclosed in quotation marks but the string "S" is. The argument must be evaluated before the F$LENGTH function can determine the length. The value represented by the symbol BUG ("BUMBLEBEE") is concatenated with the string "S"; the result is "BUMBLEBEES". The F$LENGTH function determines the length of the string "BUMBLEBEES" and returns the value 10.

- The following example uses another lexical function as the argument for the F$LENGTH function. The F$DIRECTORY function returns the name of your current default directory, including the square brackets. In the following example, the current default directory is [SALMON].

```
$ LEN = F$LENGTH(F$DIRECTORY())
$ SHOW SYMBOL LEN
  LEN = 8   Hex = 00000008  Octal = 00000000010
```

14.9.14 Using the F$DIRECTORY Lexical Function

Do not place quotation marks around the F$DIRECTORY function when it is used as an argument; the function is automatically evaluated. The result of the F$DIRECTORY function must be returned before the F$LENGTH function can determine the length. Then, the F$LENGTH function determines the length of your default directory, including the square brackets.

14.9.15 Order of Operations

An expression can contain any number of operations and comparisons. The following table lists the operators in the order in which they are evaluated if there are two or more operators in an expression. The operators are listed from highest to lowest precedence; that is, operators at the top of the table are performed before operators at the bottom.

Precedence	Operation
7	Unary plus (+) and minus (–)
6	Multiplication (*) and division (/)
5	Addition (concatenation) and subtraction (reduction)
4	All numeric and character comparisons
3	Logical .NOT. operations
2	Logical .AND. operations
1	Logical .OR. operations

If an expression contains operators that have the same order of precedence, the operations are performed from left to right.

14.9.16 Overriding Default Precedence

You can override the normal order of precedence (the order in which operation and comparison would be evaluated) by placing operations to be performed first in parentheses. Parentheses can also be nested.

14.9.17 Example

In the following example, the parentheses force the addition to be performed before the multiplication. Without the parentheses, the multiplication is performed first and the result is 26:

```
$ RESULT = 4 * (6 + 2)
$ SHOW SYMBOL RESULT
  RESULT = 32   Hex = 00000020  Octal = 00000000040
```

14.9.18 Evaluating Data Types

The result of DCL's evaluation of a symbol is either a character string or an integer value. The data type (character or integer) of a symbol is determined by the data type of the value currently assigned. The data type is not permanent: if the value changes data type, the symbol changes data type.

An expression has either an integer or a string value, depending on the types of values and the operators used.

14.9.19 Example

In the following example, the local symbol NUM is first assigned a character value and then converted to an integer value when assigned an integer expression:

```
$ NUM = "ABC"
$ NUM = 2 + 5
```

14.9.20 Values of Expressions

The following table summarizes how DCL evaluates expressions. The first column lists the different values and operators that an expression might contain. The second column tells, for each case, what the entire expression is equated to. Within the table *any value* stands for a string or an integer.

Expression	Resulting Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
+, –, or .NOT. any value	Integer
Any value .AND. or .OR. any value	Integer
String + or – string	String

Expression	Resulting Value Type
Integer + or – any value	Integer
Any value + or – integer	Integer
Any value * or / any value	Integer
Any value (string comparison) any value	Integer
Any value (numeric comparison) any value	Integer

14.10 Converting Value Types in Expressions

14.10.1 Overview

All the operands in an expression must be of the same value data type before DCL can evaluate the expression. Values either have string or integer data types. String data includes character strings, symbols with string values, and lexical functions that return string values. Integer data includes integers, symbols with integer values, and lexical functions that return integer values. When an expression contains both number and string operands, DCL either converts all strings to integers or all integers to strings.

14.10.2 Converting Expressions

In general, if you use both string and integer values, the string values are converted to integers. The only exception is when DCL performs string comparisons. In these comparisons, integers are converted to strings.

In addition, the following lexical functions let you determine or change the value of an expression:

- F$TYPE—Determines the current value type of a symbol
- F$INTEGER—Converts a string expression to an integer value
- F$STRING—Converts an integer expression to a string value

14.10.3 Converting Strings to Integers

Character strings are converted to integers in the following ways:

- Strings containing numbers are converted to their integer values. For example, the string "45" is converted to the integer 45.
- If a character string begins with T, t, Y, or y, it is converted to the integer 1.
- If a string begins with any other letter, it is converted to the integer 0.

14.10.4 Examples

The following table shows examples of strings converted to integer values:

String	Resulting Integer
‘‘123’’	123
‘‘12XY’’	0 (False)
‘‘Test’’	1 (True)
‘‘hello’’	0 (False)

14.10.5 Converting Integers to Strings

When integers are converted to character strings, the resulting string contains numbers that correspond to the integer value. The following table shows how integers are converted to string values:

Integer	Resulting String
123	‘‘123’’
1	‘‘1’’
0	‘‘0’’

14.11 Understanding Symbol Tables

14.11.1 Overview

Symbols are stored in local or global symbol tables, which are maintained by the operating system.

14.11.2 Local Symbol Tables

DCL maintains a local symbol table for your main process and for every command level that you create when you execute a command procedure, use the CALL command, or submit a batch job. When you create a local symbol, DCL places the symbol in the local symbol table for the current command level. As long as the command level is active, DCL maintains the local symbol table for that command level; when a command level is no longer active, its local symbol table (and all the symbols it contains) is deleted. See Chapter 18 for more information about processes, command procedures, and batch jobs.

14.11.3 Parameters in Local Symbol Tables

In addition to the local symbols you create, a local symbol table contains eight symbols that are maintained by DCL. These symbols, named P1, P2, and so on through P8, are used for passing parameters to a command procedure. Parameters passed to a command procedure are regarded as character strings. Otherwise, P1 to P8 are defined as null character strings (`""`). They are stored in the local symbol table.

14.11.4 Global Symbol Tables

DCL maintains only one global symbol table for the duration of a process and places all global symbols in that table. In addition to the global symbols you create, the global symbol table contains the reserved global symbols described in the following table. These global symbols give you status information on your programs and command procedures as well as on system commands and utilities.

14.11.5 $STATUS Reserved Global Symbol

$STATUS is the condition code returned by the most recently executed command. The symbol $STATUS conforms to the format of an OpenVMS operating system message code. Applications programs can set the value of the global symbol $STATUS by including a parameter value to the EXIT command. The system uses the value of $STATUS to determine which message, if any, to display and whether to continue execution at the next higher command level. The value of the lowest three bits in $STATUS is placed in the global symbol $SEVERITY.

14.11.6 $SEVERITY Reserved Global Symbol

$SEVERITY is the severity level of the condition code returned by the most recently executed command. The symbol $SEVERITY, which is equal to the lowest three bits of $STATUS, can have the following values:

0	Warning
1	Success
2	Error
3	Information
4	Severe (fatal) error

14.11.7 $RESTART Reserved Global Symbol

$RESTART has the value TRUE if a batch job was restarted after it was interrupted by a system failure. Otherwise, $RESTART has the value FALSE.

14.11.8 Symbol Table Search Order

When the command interpreter determines the value of a symbol, it searches symbol tables in the following order:

1. The local symbol table for the current command level
2. Local symbol tables for each previous command level, searching backwards from the current level
3. The global symbol table

14.12 Masking the Value of Symbols

14.12.1 SET SYMBOL Command

By default, all symbols (both global and local) defined in an outer command procedure level are accessible to inner procedure levels. However, you can isolate the local or global symbols in a command procedure from the symbols defined in other command procedures by using the SET SYMBOL command. The SET SYMBOL command masks the values of local and global symbols without deleting them. Thus, if a command procedure executes another command procedure, you can use the same symbol names in both procedures if you specify the SET SYMBOL command in the second procedure.

14.12.2 Verb String Translation

The SET SYMBOL command also controls whether DCL attempts to translate the verb string (the first word on the command line) as a symbol before processing the line. The default behavior is that the translation is attempted. The advantage to changing this behavior is that a command procedure is not affected by outer-procedure-level environments when it invokes a command.

14.12.3 Symbol Scoping State

The **symbol scope** is different for local and global symbols. When you exit a procedure level to return to a previous procedure, the symbol scoping context from the previous level is restored for both local and global symbols.

To display the current, general symbol scoping state, use the lexical function F$ENVIRONMENT("SYMBOL_SCOPE"). To display the current verb scoping state, use the lexical function F$ENVIRONMENT("VERB_SCOPE").

14.12.4 Local Symbol Scope

Local symbols are procedure-level dependent. If you define a local symbol in an outer procedure level, the symbol can be read (but not written to) at any inner procedure level. If you assign a value to a symbol that is local to an outer procedure level, a new symbol is created at the current procedure level. However, the symbol in the outer procedure level is not modified.

The SET SYMBOL/SCOPE=NOLOCAL command causes all local symbols defined at an outer procedure level to be inaccessible to the current procedure level and any inner levels. For example, if you specify SET SYMBOL/SCOPE=NOLOCAL at procedure levels 2 and 4:

- Procedure level 2 can read and write to level 2 local symbols only.
- Procedure level 3 can read (but not write to) level 2 local symbols. Level 3 can also read and write to level 3 local symbols.
- Procedure level 4 can read and write to level 4 local symbols only.

14.12.5 Global Symbol Scope

Global symbols are procedure-level independent. The current global symbol scoping context is applied subsequently to all procedure levels.

The /SCOPE=NOGLOBAL qualifier causes all global symbols to become inaccessible for all subsequent commands until either the /SCOPE=GLOBAL qualifier is specified or the procedure exits to a previous level at which global symbols were accessible. In addition, specifying the /SCOPE=NOGLOBAL qualifier prevents you from creating any new global symbols until the /SCOPE=GLOBAL qualifier is specified.

14.13 Understanding Symbol Substitution

14.13.1 Overview

In certain contexts, DCL uses a string of characters beginning with a letter as a symbol name or a lexical function. In these contexts, DCL tries to replace the symbol or lexical function with its value. Replacing a symbol with its current value is referred to as symbol substitution. If you use a symbol or lexical function in any other context, you must use a substitution operator to request symbol substitution.

14.13.2 Automatic Evaluation of Symbols

DCL automatically evaluates symbols and lexical functions when they are used as follows:

- On the right side of an assignment (=) statement
- In an argument for a lexical function
- In a DEPOSIT, EXAMINE, IF, or WRITE command
- At the beginning of a command line when the string is not followed by an equal sign or a colon
- In the brackets on the left side of an assignment statement when you are performing substring substitution or numeric overlays (see Section 14.7.13)

14.13.3 Examples

In the following examples, the command interpreter uses any character string beginning with an alphabetic character as a symbol name and any string beginning with a number or with the radix operator (%) as a literal numeric value.

- In the following example, COUNT is automatically recognized and evaluated as a symbol:

```
$ TOTAL = COUNT + 1
```

- In the second line of this example, the symbol QUERY is automatically evaluated when it is used with the F$LENGTH function. In addition, the F$LENGTH function is automatically evaluated because it is on the right side of an assignment statement:

```
$ QUERY = "Have we met before?"
$ LEN = F$LENGTH(QUERY) + 5
$ SHOW SYMBOL LEN
  LEN = 27   Hex = 0000001B  Octal = 000033
```

- In the following example, the IF command uses both A and B as symbol names and uses their current values:

```
$ IF A .EQ. B THEN WRITE SYS$OUTPUT "DONE"
```

- In the second line of this example, the command interpreter automatically replaces PDEL with its current value and executes the resulting command:

```
$ PDEL = "DELETE SYS$PRINT/ENTRY="
$ PDEL 181
```

- In the following example, DCL automatically defines the symbol BELL as the value of 7 and then assigns a new value based on the bracketed values on the left side of the assignment statement.

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
 BELL = 39   Hex = 00000027   Octal = 00000000047
```

14.13.4 Forced Symbol Substitution

To force substitution of a symbol that is not in one of the positions listed, enclose the symbol with apostrophes ('), as follows:

```
$ TYPE 'B'
```

To force substitution of a symbol within a quoted character string, precede that symbol with two apostrophes (') and follow it with a single apostrophe (') as follows:

```
$ T = "TYPE ''B'"
```

14.13.5 Precedence of Symbol Substitution

When processing a command line, DCL replaces symbols with their values in the following order:

- Forced substitution

  From left to right, DCL replaces all strings delimited by apostrophes (or double apostrophes for strings within quotation marks). Symbols preceded by single apostrophes are translated iteratively; symbols preceded by double apostrophes are not.

- Automatic substitution

  From left to right, DCL evaluates each value in the command line, executing it if it is a command and evaluating it if it is an expression. Symbols in expressions are replaced by their assigned values; this substitution is not iterative.

14.13.6 Example

The following example demonstrates the effect of the order in which DCL substitutes symbols. First, the symbols PN, FILE1, and NUM are defined:

```
$ PN = "PRINT/NOTIFY"
$ FILE1 = "[BOLIVAR]TEST_CASE.TXT"
$ NUM = 1
```

Given the preceding symbol definitions, the following commands print the file named [BOLIVAR]TEST_CASE.TXT:

```
$ FILE = "'FILE''NUM'"
$ PN 'FILE'
```

In the first command, forced substitution causes NUM to become 1, making FILE''NUM' become FILE1. If you enter the command SHOW SYMBOL FILE, you see that FILE = " 'FILE1' ".

The second command performs two substitutions. First, 'FILE' is substituted with 'FILE1'. 'FILE1' also requires substitution because it is enclosed in apostrophes ('). Automatic substitution causes FILE1 to become [BOLIVAR]TEST_CASE.TXT. This file name is then appended to the value of PN, which is PRINT /NOTIFY. The resulting string is as follows:

```
$ PRINT/NOTIFY [BOLIVAR]TEST_CASE.TXT
```

14.13.7 Symbol Substitution Operators

You can use a substitution operator to request symbol substitution in places where DCL does not usually perform it. DCL accepts two substitution operators:

- Apostrophe (')
- Ampersand (&)

The difference between these two operators is the time when the substitution occurs. Symbols preceded by apostrophes are substituted during the first phase of DCL command processing; symbols preceded by ampersands are substituted during the second phase. For more information on the phases of command processing, see Section 14.14.

14.13.8 The Apostrophe (')

The apostrophe (') is the most frequently used substitution operator. Use it to request symbol substitution when you use a symbol in place of a command parameter or qualifier. Use the apostrophes to request symbol substitution on the right side of a string assignment (:=) statement.

To request symbol substitution within a quoted character string, place two apostrophes before the symbol name and one apostrophe after it.

When you use apostrophes to request symbol substitution, you cannot continue the line (with the hyphen continuation character) in the middle of the value that is being substituted.

14.13.9 Examples

- In the following example, the TYPE command requires a file specification. The apostrophes indicate that LIT is a symbol that must be evaluated. If you omit the apostrophes, DCL looks for a file called LIT.LIS (.LIS is the default file type for the TYPE command):

```
$ LIT = "LIGHT.BILLS"
$ TYPE 'LIT'
```

- In the following example, the value for NAME is substituted so that FILE becomes REPORT.DAT:

```
$ NAME := REPORT
$ FILE := 'NAME'.DAT
$ SHOW SYMBOL FILE
  FILE = "REPORT.DAT"
```

- In the following example, the current value of the symbol NAME is FRED:

```
$ MESSAGE = "Creating file ''NAME'.DAT"
```

  Therefore, MESSAGE has the following value:

```
Creating file FRED.DAT
```

14.13.10 The Ampersand (&)

The ampersand (&) is also a substitution operator that the command interpreter recognizes. In many cases, the apostrophe and the ampersand perform the same function. Ampersands are most effective as substitution operators when they are used with apostrophes to affect the order in which substitution is performed.

The action the command interpreter takes when a symbol is undefined depends on the context of the command. For more information, see Section 14.14.14.

14.13.11 Examples

- In the first command shown here, the command interpreter replaces the symbol NAME with its current value during the first phase of command processing (scanning). The second command replaces the symbol NAME with its current value during the second phase of command processing (parsing). The result is the same, even though the methods are different:

```
$ TYPE 'NAME'
$ TYPE &NAME
```

- In the following example, the ampersand (&) is used with apostrophes to affect the substitution order:

```
$ P1 = "FRED.DAT"
$ COUNT = 1
$ TYPE &P'COUNT'
```

  First, the command interpreter evaluates the symbol enclosed by apostrophes ('COUNT'). The result is as follows:

```
TYPE &P1
```

Second, the command interpreter evaluates the symbol preceded by an ampersand (P1). The result is as follows:

```
TYPE FRED.DAT
```

- In the following example, apostrophes are used with both P and COUNT:

  ```
  $ TYPE 'P''COUNT'
  ```

  Working left to right, the command interpreter attempts to evaluate P. Because P is not a defined symbol, DCL gives it a null value. Next, it evaluates the symbol COUNT. The result is as follows:

  ```
  TYPE 1
  ```

- In the following example, A is equated to the current value of B:

  ```
  $ B = "MYFILE.DAT"
  $ A = "&B"
  $ TYPE 'A'
  ```

  The ampersand (&) does not cause symbol substitution when it is used inside quotation marks (" "). Therefore, when the assignment is made, the value of B is not substituted. However, the TYPE command displays MYFILE.DAT. This occurs because the command interpreter first substitutes the value &B for A. Next, it substitutes MYFILE.DAT for the symbol &B. If you were to redefine B, the result of the TYPE command would change accordingly.

14.13.12 Rules for Using Ampersands

Observe the following rules:

- Place the ampersand before, but not after, the symbol name.
- An ampersand must follow a delimiter (any blank or special character).
- You cannot use ampersands to request substitution within character strings enclosed in quotation marks (" ").
- You cannot use ampersands to concatenate two or more symbol names.
- In general, do not use the ampersand for symbol substitution unless it is required to translate your symbols correctly.

14.14 The Three Phases of Command Processing

14.14.1 Overview

The command interpreter performs symbol substitution in three phases.

14.14.2 Phase 1: Command Input Scanning

In command input scanning (also called the lexical input phase), the command interpreter evaluates symbols preceded by apostrophes from left to right. Symbols that are preceded by single apostrophes are translated iteratively, as described in Section 14.14.8. Symbols preceded by two apostrophes are not translated iteratively.

14.14.3 Phase 2: Command Parsing

In the command parsing phase:

- The command interpreter analyzes the command line. It checks the first item on the line to see if it is a symbol. If it is, it is evaluated.
- The command interpreter evaluates symbols preceded by ampersands from left to right.

Symbol substitution during this phase is not iterative.

14.14.4 Phase 3: Expression Evaluation

During the expression evaluation phase:

- The command interpreter evaluates symbols that are preceded by the DEPOSIT, EXAMINE, IF, and WRITE commands.
- The command interpreter evaluates symbols within lexical functions.

Symbol substitution during this phase is not iterative.

14.14.5 Symbol Substitution on Data Lines

Note that the command interpreter does not scan any lines that are read as input data by commands or programs executed within a command procedure. Therefore, the command interpreter does not perform symbol substitution within these data lines.

14.14.6 Example

In the following example, the program AVERAGE reads 55, 57, and 9999 from SYS$INPUT (the command input stream). These data lines are never read by the command interpreter. If you enter symbol names as input, they are not evaluated:

```
$ RUN AVERAGE
55
57
9999
```

14.14.7 Repetitive and Iterative Substitution

Symbol substitution can be repetitive or iterative:

- Repetitive substitution results when more than one type of substitution occurs in a single command line.
- Iterative substitution occurs when the command interpreter examines a substituted value to see if the value itself is a symbol. Iterative substitution occurs only when symbols preceded by apostrophes are translated during the first phase of command processing.

14.14.8 Phase 1 Substitutions

When you use an apostrophe (') to request symbol substitution, the command interpreter performs iterative substitution during the first phase of command processing.

Substitution using apostrophes is not iterative when a symbol is included in a quoted character string.

14.14.9 Example

In the following example, the substitution is not iterative:

```
$ MAC = "5"
$ A = "''MAC'"
$ B = "''A'"
$ SHOW SYMBOL B
  B = 5  Hex = 00000005  Octal = 00000000005
```

After the statement B = 'A' the resulting value of the symbol B is 5 because:

- The symbol name A is enclosed in apostrophes, so it is replaced with its current value ('MAC').
- Because this value ('MAC') is also enclosed in apostrophes, the command interpreter replaces MAC with its current value (5).
- Because this value (5) has no apostrophes, the first phase of command processing is complete. No further substitution is required during the second or third phases. Therefore, 5 is the final value given to the symbol name B.

Note, however, what happens when you include A in a quoted character string:

```
$ B = "''A'"
$ SHOW SYMBOL B
  B = "'MAC'"
```

In this case, B has the value 'MAC'. The symbol name A is replaced only once because substitution is not iterative within quoted character strings.

14.14.10 Phase 2 Substitution

The command interpreter performs iterative substitution automatically only when an apostrophe is in the command line. In some cases, you may want to nest command synonym definitions.

14.14.11 Example

In the following example, when EXEC is processed, the command interpreter performs substitution only once:

```
$ MAC = "TYPE A.B"
$ EXEC = "'MAC'"
$ EXEC
```

The result is the string 'MAC'. The command interpreter displays an error message because it does not recognize MAC as a command. This error occurs because during the first phase of command processing, no substitution is performed (the string EXEC is not delimited by apostrophes). During the second phase,

the string 'MAC' is substituted for EXEC because EXEC is the first value on the command line. This substitution is not iterative. Therefore, even though 'MAC' is delimited by apostrophes, no additional substitution is performed.

To use the command synonym EXEC correctly, enclose it in apostrophes:

```
$ 'EXEC'
```

In this case, the symbol EXEC is evaluated during the first phase of command processing. Because this substitution is iterative, ('MAC') is also evaluated and the string TYPE A.B is substituted.

14.14.12 Phase 3 Substitution

When the command interpreter analyzes an expression in a command, any symbols specified in the expression are replaced only once. You can, however, force iterative substitution by using an apostrophe or an ampersand in the expression. When you force iteration in this way, you must remember the following:

- The command interpreter performs all substitutions requested by apostrophes and ampersands before the command string is executed.
- Commands that automatically perform symbol substitution do so after the first and second phases of command processing.

Note, however, that if substitution does not result in a valid symbol name, the command fails.

14.14.13 Examples

- The following example shows iterative substitution in an IF command:

  ```
  $ P1 = "FRED.DAT"
  $ COUNT = 1
  $ IF P'COUNT' .EQS. "" THEN GOTO END
  ```

  When the command interpreter scans this line, it replaces the symbol COUNT with its current value. The result is as follows:

  ```
  IF P1 .EQS. "" THEN GOTO END
  ```

  Because this string has no apostrophes, the command interpreter does not perform any more substitution. However, when the IF command executes, it automatically evaluates the symbol name P1 and replaces it with its current value.

- In the following example, the symbol name FILENAME is invalid:

  ```
  $ FILENAME = "A.B"
  $ IF 'FILENAME' .NES. "" THEN TYPE 'FILENAME'
  ```

The command interpreter replaces the symbol FILENAME with its current value (A.B). The result is as follows:

```
IF A.B .NES.  "" THEN TYPE A.B
```

When the IF command executes the command line, A.B is not a valid symbol and an error occurs. For this IF command to be processed correctly, omit the apostrophes, as follows:

```
$ IF FILENAME .NES. "" THEN TYPE 'FILENAME'
```

14.14.14 Undefined Symbols

If a symbol is not defined when it is used in a command line, the command interpreter either displays an error message or replaces the symbol with a null string, depending on the context. The rules are as follows:

- During the first and second phases of command processing, the command interpreter replaces all undefined symbols that are preceded by apostrophes or ampersands with null strings.
- During the third phase of command processing, if the command interpreter finds an undefined symbol, it displays a warning message and does not finish processing.

14.14.15 Examples

- The following example shows how the command interpreter processes an undefined symbol that is preceded by an apostrophe:

  ```
  $ FILE := MYFILE'FILE_TYPE'
  $ SHOW SYMBOL FILE
    FILE = "MYFILE"
  $ PRINT 'FILE'
  ```

  When the symbol FILE is created, the symbol FILE_TYPE is replaced with its current value. If FILE_TYPE is not defined, the command interpreter replaces FILE_TYPE with a null string. The absence of a file type in the file specification causes the PRINT command to use the default file type .LIS. Thus, the file specification is interpreted as MYFILE.LIS.

- In the following example, the expression is evaluated during the third phase of command processing:

  ```
  $ A = 1
  $ C = A + B
  %DCL-W-UNDSYM, undefined symbol - check validity and spelling
  ```

  The symbol B is undefined, so the command interpreter cannot evaluate the expression.

14.15 An Alternative to Using Symbols: Automatic Foreign Commands

14.15.1 Overview

You can also invoke a command procedure (.COM file type) or executable image (.EXE file type) from DCL level without defining a symbol for that procedure. Using automatic foreign commands, DCL can search a specific set of directories for a command procedure or executable image and run it automatically.

14.15.2 How Automatic Foreign Commands Work

When you enter a command verb that is not a DCL symbol and that is not in the DCL command tables, the system usually displays the following message:

```
DCL-W-IVVERB, unrecognized command verb - check validity and spelling
```

However, if the logical name DCL$PATH is defined (and is not blank), DCL instead performs an RMS $SEARCH for any file that contains the invalid verb in its file name and DCL$PATH:.* as the default file specification.

If DCL finds a .COM or .EXE file, DCL will automatically execute that file with the rest of the command line as its parameters. (This behavior is similar to the PATH options found in DOS, UNIX, and other operating systems.)

14.15.3 Example

In the following example, the DCL symbol SYSGEN is no longer needed. DCL looks in the SYS$SYSTEM directory and finds SYSGEN.EXE. DCL acts like the symbol "SYSGEN" was defined as "SYSSYSTEM:SYSGEN" which causes the SYSGEN image to be activated as a foreign command.

```
$ SYSGEN
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \SYSGEN\
$ DEFINE DCL$PATH SYS$SYSTEM,SYS$DISK:[]FOO
$ SYSGEN SHOW MAXPROCESSCNT
Parameter Name   Current   Default    Min.    Max.    Unit  Dynamic
--------------   -------   -------   ------- -------  ----  -------
MAXPROCESSCNT        157        32        12    8192 Processes
```

14.15.4 Example

In the following example, SS does not need to the defined as "@SS.COM" because DCL will automatically search the SYS$SYSTEM directory for SS.COM or SS.EXE. If that fails, DCL will search the current directory for SS.COM or SS.EXE.

```
$ TYPE SS.COM
$ SHOW SYMBOL/LOCAL/ALL
$ EXIT
$ SS "This is a parameter"
  P1 = "This is a parameter"
  P2 = ""
  P3 = ""
  P4 = ""
  P5 = ""
  P6 = ""
  P7 = ""
  P8 = ""
$ SS.EXE "This is a parameter"
  P1 = ".EXE"
  P2 = "This is a parameter"
  P3 = ""
  P4 = ""
  P5 = ""
  P6 = ""
  P7 = ""
  P8 = ""
```

In the example, DCL locates SS.COM and acts like "SS" had been a symbol defined as "@SS.COM". The command procedure is activated with the rest of the command line parsed as parameters. Note that "SS.EXE" does not invoke the image SS.EXE, but instead invokes SS.COM with two parameters, the first being the text string ".EXE". This is consistent with the way command parsing and symbol substitution is performed by the OpenVMS operating system.

14.15.5 Using Automatic Foreign Commands

Note the following:

- The logical name DCL$PATH can be a search-list type logical.
- Only the node, device, and directory portions of each translation of the logical name are used.
- Normal logical precedence takes place. Users can override a system definition of DCL$PATH by defining their own. If a system definition exists and the user does not want the feature, it can be turned off by overriding the logical with a definition of " ".
- The set of valid characters for DCL verbs and symbol names differs from the set of valid characters for filenames. For example, DCL symbols cannot contain a hyphen (-) or start with a dollar sign ($). If the image or procedure you wish to execute is not valid as a DCL symbol name, it cannot be directly invoked by this new feature.
- DCL has not parsed the command. It is up to the image being invoked to perform its own command parsing. For C programs, use the "argc" and "argv" parameters to the main( ) routine. For programs written in other languages, call LIB$GET_FOREIGN to obtain the entire command line, which must then be parsed by the program.

- If a directory contains both a command procedure and an executable image, whichever file is found first will be invoked. On OpenVMS systems, directories are in alphabetical order, so a ".COM" file will be found before a ".EXE" file. A network file specification in the DCL$PATH logical pointing to a node running some other operating systems could result in a ".EXE" file being found before a ".COM" file.

  Because DCL performs the search with the invalid verb as the file specification and "DCL$PATH:.*" as the default file specification, it is possible to define a logical in such a way that a specific file is found. For example, if you define the logical FOO to be "FOO.EXE", and type "FOO" at the DCL prompt, you will never invoke FOO.COM, only FOO.EXE.

14.15.6 Caution: Defining DCL$PATH

If you are a privileged user and set your default device and directory to other user accounts, you should not place "SYS$DISK:[ ]" in the definition of the DCL$PATH logical name. Doing so will cause DCL to search the current directory, where a typographical error or poor placement of the translation within the search list could cause user images in the current directory to be found and mistakenly invoked with privileges.

14.15.7 Restrictions

Note the following restrictions:

- You cannot use automatic foreign commands on any versions of the OpenVMS operating system prior to Version 6.2.
- Because new verbs can be added to the DCL command table at any time, a command that works with automatic foreign commands one day may not work at a later date.
- The automatic foreign commands feature does not work in all cases. In the following example, DCL (which looks only at the first four characters of any verb) finds a match with the SHOW verb (the first four letters of SHOWME) and executes the SHOW USERS command instead of the SHOWME.COM procedure. If you defined SHOWME as a DCL symbol, then the SHOWME command would invoke SHOWME.COM.

```
$ DEFINE DCL$PATH SYS$SYSTEM,SYS$DISK:[]FOO
$ TYPE SHOWME.COM
$ SHOW SYMBOL P1
$ EXIT
$ SHOWME USERS
      OpenVMS User Processes at MARCH 2, 1995 01:40 PM
    Total number of users = 1,  number of processes = 11

 Username     Interactive  Subprocess   Batch
 RSMITH             9         2
```

15

Introduction to Command Procedures: Programming with DCL

15.1 Overview

Command procedures are files that contain DCL commands and data lines used by DCL commands. This chapter is divided into major sections that include the following:

- Basic information about writing command procedures
- Step-by-step procedure for writing command procedures
- Executing command procedures
- Exiting, interrupting, and error handling command procedures
- Login command procedures

15.1.1 Resources

Refer to the *OpenVMS DCL Dictionary* for complete information on all commands described in this chapter.

15.1.2 Types of Command Procedures

There are two types of DCL command procedures:

- Simple

  Execute a series of DCL commands in the order in which they are written

- Complex

  Perform program-like functions

Basic Information for Writing Command Procedures

15.2 Overview

15.2.1 Creating Command Procedures

There are two ways to create command procedures:

- Use a text editor such as EVE or EDT to create a new file
- Use the DCL command CREATE to create a new file

The file that you create can contain command lines, labels, comments, conditional statements, and variables.

15.2.2 File Type

The default file type for command procedures is .COM. If you specify the .COM file type when you name a command procedure, you can execute the procedure by specifying the file name only. The SUBMIT and execute procedure (@) commands assume the file type is .COM unless you specify otherwise.

15.2.3 Writing Commands

The following are suggestions for including commands in command procedures:

- Use complete names for commands and qualifiers. This will help to ensure that your command procedure is upwardly compatible to future releases of OpenVMS.
- Use continuation lines to make a procedure easier to read. Note that continuation lines do not begin with dollar signs. For example:

```
$ PRINT LAB.DAT   -
     /AFTER=17:00 -
     /COPIES=20   -
     /NAME="COMGUIDE"
```

15.2.4 Writing Command Lines

When writing command lines:

- You must use a dollar sign ($) to begin each line containing a command, comment, or label.
- If you want to include a line containing data, omit the dollar sign ($) on that line.

- If you need to include a data line that begins with a dollar sign ($), use the DCL commands DECK and EOD. For example:

```
$ ! Everything between the commands DECK and EOD
$ ! is written to the file WEATHER.COM
$ !
$ CREATE WEATHER.COM
$ DECK
$ FORTRAN SUMMER
$ LINK SUMMER
$ RUN SUMMER
$ EOD
$ !
$ ! Now execute WEATHER.COM
$ @WEATHER
$ EXIT
```

15.2.5 Use of Dollar Signs ($)

Note that command lines that do not begin with a dollar sign *might* be correctly interpreted by DCL, but Digital strongly recommends that any DCL command line start with a dollar sign.

15.3 Use of Labels

15.3.1 Rules for Using Labels

Labels are used in DCL command procedures to mark the beginning of loops, sections of code, or subroutines. Note the following rules when using labels:

- Put labels on separate lines to make loops, subroutines, and conditional code more visible.
- Use label names that contain fewer than 255 characters and no blank spaces.
- Differentiate labels from commands by placing labels immediately after the dollar sign ($) and by preceding commands with spaces.
- End each label with a colon.
- You cannot delete labels.

15.3.2 Labels in Local Symbol Tables

As the command interpreter encounters labels, it enters them in a special section of the local symbol table. The amount of space available for labels is limited. If a command procedure uses many symbols and contains many labels, the command interpreter might run out of symbol table space and issue an error message. If this occurs, include the DELETE/SYMBOL command in your procedure to delete symbols as they are no longer needed. (Note, however, that you cannot delete labels.)

15.3.3 Duplicate Labels

If a command procedure uses the same label more than once, the new definition replaces the existing one in the local symbol table.

When duplicate labels exist, the GOTO command transfers control to the label that DCL has processed most recently. The GOTO command also uses the following rules when processing duplicate labels:

- If all duplicate labels precede the GOTO command, control transfers to the label nearest the GOTO command.
- If duplicate labels precede and follow the GOTO command, control transfers to the preceding label nearest the GOTO command.
- If all duplicate labels follow the GOTO command, control transfers to the label nearest the GOTO command.

15.4 Using Comments in Command Procedures

15.4.1 Suggested Uses of Comments

It is good programming practice to include comments in command procedures. Comments can be helpful when updating or troubleshooting the command procedure. Comments can be used as follows:

- At the beginning of a procedure to describe the procedure and the parameters passed to it.
- At the beginning of each block of commands to describe that section of the procedure.
- To separate command sequences with lines containing both a dollar sign and an exclamation point ($!). This makes it easier to see the outline of the command procedure. If you insert blank lines, the command interpreter interprets them as data lines and produces a message warning you that the data lines were ignored.

15.4.2 Rules for Using Comments

The following rules apply when writing comments in command procedures:

- Use an exclamation point (!) to indicate the beginning of a comment; the command interpreter ignores all text to the right of an exclamation point when the command procedure executes.
- To include a literal exclamation point in a command line, enclose the exclamation point in quotation marks (`" "`).

How to Write Command Procedures

15.5 Overview

15.5.1 Before You Begin

Before you begin writing a command procedure, perform the tasks interactively that the command procedure will execute. As you type the necessary commands, note any variables and conditionals that are used, and any iterations that occur.

15.5.2 Definitions

- **Variable**

  Data that changes each time you perform a task.

- **Conditional**

  Any command or set of commands that can vary and therefore must be tested each time you perform the task.

- **Iteration**

  Any command or set of commands that are performed repetitively until a condition is met.

15.5.3 About This Section

This section contains the steps to write a simple command procedure. The example used throughout this section is a command procedure called CLEANUP.COM. This procedure can be used to clean up a directory.

This section also includes the completed command procedure, CLEANUP.COM.

15.5.4 The Steps for Writing Command Procedures

Follow these steps to write a command procedure:

Step	Task
1	Design the command procedure.
2	Assign variables and test conditionals.
3	Add loops.
4	End the command procedure.
5	Test and debug the program logic.
6	Add clean up tasks.
7	Finish the procedure.

15.6 Step 1: Design the Command Procedure

15.6.1 How to Design Command Procedures

Follow these steps to design a command procedure:

Step	Task
1	Decide which tasks your procedure will perform.
2	Determine any variables your command procedure will use and how they will be loaded.
3	Determine what conditionals the command procedure requires and how you will test them.
4	Decide how you will exit from the command procedure.

15.6.2 Deciding Which Tasks to Perform

There are certain commands that are usually executed during clean up operations. The following table lists those commands and the tasks that they perform:

Command	Task Performed
DIRECTORY	Displays the contents of the current directory
TYPE filespec	Displays a file
PURGE filespec	Purges a file
DELETE filespec	Deletes a file
COPY filespec new-filespec	Copies a file

15.6.3 Variables

Any data that changes when you perform a task is a variable. If you create or delete files in your directory, the file names will be different each time you clean your directory; therefore, the file names in CLEANUP.COM are variables.

15.6.4 Conditionals

Any command that must be tested each time you execute a command procedure is considered conditional. Because any or all of the commands in CLEANUP.COM might be executed, depending on the operation you need to perform, each command is conditional.

15.6.5 Design Decisions

After you have determined what variables and conditionals you will use in the CLEANUP.COM command procedure, you must decide how to load the variables, test the conditionals and exit from the command procedure. For the CLEANUP.COM command procedure, the following decisions have been made:

Task	How Accomplished
Load variables	The command procedure gets the file names from the terminal.
Test Conditionals	The command procedure: • Gets a command name from the terminal and executes the appropriate statements based on the command name. • Ensures that the first two characters of each command name are read to differentiate between the DELETE and DIRECTORY commands.
Exit from loop	You must enter the EXIT command to exit from the loop.

15.6.6 Ordering Commands

To make command procedures easier to understand and maintain, write statements so the procedures execute from the first command to the last command.

15.7 Step 2: Assign Variables and Test Conditionals

15.7.1 Overview

There are many ways to assign values to variables. In this section, we will discuss using the INQUIRE command. For additional methods, see Chapter 16.

15.7.2 How to Assign Variables and Test Conditionals

Follow these steps to assign values to variables and test conditionals:

Step	Task
1	Assign values to variables using the INQUIRE command.
2	Determine which action should be taken.
3	Test the conditional using IF and THEN statements.
4	Write program stubs and insert them into the command procedure as placeholders for commands.
5	Write error messages, if necessary.

15.7.3 Using the INQUIRE Command

The INQUIRE command prompts for a value, reads the value from the terminal, and assigns the value to a symbol.

By default, the INQUIRE command:

- Converts responses to uppercase
- Replaces multiple blanks and tabs with a single space

- Removes leading and trailing spaces
- Performs apostrophe substitutions if the response includes symbols or lexical functions

15.7.4 Preserving Literal Characters

To preserve lowercase characters, multiple spaces and tabs when using the INQUIRE command, enclose your response in quotation marks (" "). To include quotation marks in your response, enclose the quoted text in quotation marks (""text"").

15.7.5 Example: Using the INQUIRE Command

```
$ INQUIRE COMMAND-
  "Enter command (DELETE, DIRECTORY, PRINT, PURGE, TYPE)"
```

This command line is used in CLEANUP.COM to prompt the user for a command name. The INQUIRE command equates the value entered to the symbol COMMAND.

15.7.6 Testing Conditionals

After the INQUIRE command prompts for a variable, the command procedure must include a statement that determines what action is to be taken. For example, to determine which command to execute, you must include statements in the command procedure that check the command entered by the user against each possible command.

15.7.7 IF... THEN Commands

To test whether a condition is true, use the IF and THEN commands. The following table shows the possibilities that you must check for in CLEANUP.COM:

If...	Then...
a match is found,	execute the command.
a match is not found,	go on to the next command.
no match is found after all valid commands have been checked,	output an error message.

15.7.8 Writing Program Stubs

A **program stub** is a temporary section of code that you use in your procedure while you test the design. Usually, a program stub outputs a message stating the function that it is replacing. After the overall design works correctly, replace each stub with the correct coding.

15.7.9 Example: Assigning Variables and Testing Conditionals

The following example show how to assign variables and test conditionals:

```
$ INQUIRE COMMAND-
  "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
$ IF COMMAND .EQS. "EXIT" THEN EXIT
$!
$! Execute if user entered DELETE
$ DELETE:
$    IF COMMAND .NES "DELETE" THEN GOTO DIRECTORY          ❶ ❷
$    WRITE SYS$OUTPUT "This is the DELETE section."        ❸
$! Execute if user entered DIRECTORY
$ DIRECTORY:                                               ❹
$    IF COMMAND .NES "DIRECTORY" THEN GOTO PRINT
$    WRITE SYS$OUTPUT "This is the DIRECTORY section."
   .
   .
   .
$! Execute if user entered TYPE
$ TYPE:
$    IF COMMAND .NES "TYPE" THEN GOTO ERROR                ❺
$    WRITE SYS$OUTPUT "This is the TYPE section."
$!
$ ERROR:
$    WRITE SYS$OUTPUT "You have entered an invalid command." ❻
$!
$ EXIT
```

As you examine the example, note the following:

❶ This IF statement tests to see if the command that the user entered (COMMAND) is equal to "DELETE". If COMMAND is equal to DELETE, then the command procedure executes the next command.

❷ This statement also includes a GOTO command. A GOTO command is used to change the flow of execution to a label in the procedure. In this case, the procedure will go to the DIRECTORY label if COMMAND is not equal to DELETE.

❸ This statement is a program stub. After the logic of the command procedure is tested, this line will be replaced with the actual commands required for a DELETE operation.

❹ This is the label for the DIRECTORY subroutine. Note that the labels that identify each command block are the same as the commands on the option list. This allows you to use the symbol COMMAND (which is equated to the user's request) in the GOTO statement.

❺ This IF statement tests to see if the "TYPE" command was entered. If "TYPE" was entered, the procedure will output "This is the TYPE section." However, because this is the last command you will be testing for, if the command entered is not "TYPE," the program will display an error message.

❻ If all commands have been tested and no valid command name is found, then the program will output, "You have entered an invalid command."

15.8 Step 3: Add Loops

15.8.1 Overview

A loop is a group of statements that execute repeatedly until a condition is met. A Loop works as follows:

1. Obtains a value from user input
2. Processes the command
3. Repeats the process until the user exits the command procedure

15.8.2 Writing a Loop

To write a loop follow this procedure:

Step	Action
1	Begin the loop with a label.
2	Test a variable to determine whether you need to execute the commands in the loop.
3	If you do not need to execute the loop, go to the end of the loop.
4	If you need to execute the loop, perform the commands in the body of the loop, then return to the beginning of the loop.
5	End the loop.

15.8.3 Example: GET_COM_LOOP

The following example shows the usage of loops in the CLEANUP.COM command procedure:

```
$ GET_COM_LOOP:
$     INQUIRE COMMAND-
$     "Enter command (DELETE, DIRECTORY. EXIT, PRINT, PURGE, TYPE)"
$     IF COMMAND .EQS. "EXIT" THEN GOTO END_LOOP
$!
$! Execute if user entered DELETE
$ DELETE:
$     IF COMMAND .NES. "DELETE" THEN GOTO DIRECTORY
$     WRITE SYS$OUTPUT "This is the DELETE section."
$     GOTO GET_COM_LOOP
   .
   .
   .
$ END_LOOP:
$     WRITE SYS$OUTPUT "Directory ''F$DIRECTORY()' has been cleaned"
$ EXIT
```

Once a command executes, control is passed back to the GET_ COM_LOOP label until a user enters the EXIT command. When an EXIT command is entered, the procedure outputs a message stating that the directory has been cleaned.

15.9 Step 4: End the Command Procedure

15.9.1 How to End Command Procedures

To end a command procedure, follow this procedure:

Step	Action
1	Decide where you might need to exit or quit from the command procedure.
2	Place EXIT or STOP commands as appropriate.

15.9.2 Using the EXIT Command

You can put an EXIT command in your command procedure to:

- Ensure that a procedure does not execute certain lines
- End procedures that have more than one execution path
- End a command procedure

15.9.3 Examples

- The following is an example of using an EXIT command to avoid executing an error handling routine that is located at the end of a procedure:

```
   .
   .
   .
$ EXIT ! End of normal execution path
$ ERROR_ROUTINE
   .
   .
   .
```

- The following is an example of using the EXIT command to end a procedure that has more than one execution path:

```
$ START:
$     IF P1 .EQS. "TAPE" .OR. P1 .EQS. "DISK" THEN GOTO 'P1'
$     INQUIRE P1 "Enter device (TAPE or DISK)"
$     GOTO START
$ TAPE: !Process tape files
   .
   .
   .
$     EXIT
$ DISK: ! Process disk files
   .
   .
   .
$     EXIT
```

The commands following each of the labels (TAPE and DISK) provide different paths though the procedure. The EXIT command before the DISK label ensures that the commands after the DISK label do not execute unless the procedure branches explicitly to the label.

15.9.4 When to Use the EXIT Command

The EXIT command is not required at the end of procedures because the end-of-file of the procedure causes an implicit EXIT command. However, Digital recommends use of the EXIT command.

15.9.5 Using the STOP Command

You can use the STOP command in a command procedure to ensure that the procedure terminates if a severe error occurs. If the STOP command is in a command procedure that is executed interactively, control is returned to the DCL level. If a command procedure is being executed in batch mode, the batch job terminates.

15.9.6 Example

This command line tells the procedure to stop if a severe error occurs:

```
$ ON SEVERE_ERROR THEN STOP
```

15.10 Step 5: Test and Debug the Program Logic

15.10.1 What Needs to Be Tested?

Once you have written the code using program stubs, you should test the overall logic of the command procedure. You should test all possible paths of execution.

15.10.2 How to Test and Debug Command Procedures

Follow this procedure to test and debug command procedures:

Step	Action
1	Test the program logic by entering each valid command in the command procedure.
2	Continue testing the program logic by entering an invalid command.
3	Finish testing the program logic by exiting from the command procedure using the EXIT command.
4	If necessary, debug the program using the SET VERIFY, SET PREFIX, or SHOW SYMBOL commands.

15.10.3 Example: Testing the Program Logic

The following example shows how to test the command procedure by entering and executing every possible command, an invalid command, and then exiting:

```
$ @CLEANUP
 Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): DELETE
 This is the DELETE section.
 Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): DIRECTORY
 This is the DIRECTORY section.
 .
 .
 .
 Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): PRINF
 You have entered an invalid command.
 Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): EXIT
$
```

15.10.4 Debugging Command Procedures

The following commands can be used to help debug command procedures:

- SET VERIFY

  Displays each line before it is executed. When an error occurs with verification set, you see the error and the line that generated the error. You can use keywords with the SET VERIFY command to indicate that only command lines or data lines are to be verified.

  The SET VERIFY command remains in effect until you log out, you enter the SET NOVERIFY command, or you use the F$VERIFY lexical function to change the verification setting. (Chapter 17 contains more information on changing verification settings.)

- SET PREFIX

  If verification is in effect, you can also use the DCL command SET PREFIX to time-stamp a procedure log file by prefixing each command line with the time it is executed.

- SHOW SYMBOL

  The SHOW SYMBOL command can be used to determine how symbols in the procedure are defined.

15.10.5 Example: Debugging Using the SET VERIFY Command

In the following example, the label END_LOP is spelled incorrectly. You can see exactly where the error is because verification is turned on:

```
$ SET VERIFY
$ @CLEAN
$ GET_COM_LOOP:
$    INQUIRE COMMAND -
     "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
 Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): EXIT
$    IF COMMAND .EQS. "EXIT" THEN GOTO END_LOP
 %DCL-W-USGOTO, target of GOTO not found -
 check spelling and presence of label
```

To correct the error, change the label to END_LOOP.

15.10.6 Example: Debugging Using the SET PREFIX Command

The following example illustrates the use of time-stamping:

```
$  SET VERIFY
$  @TEST
$  SET DEFAULT SYS$LOGIN
$  SHOW DEFAULT
   USER$:[SMYTHE]
$  SET PREFIX "(!5%T) "
$  @TEST
 (17:52)  $ SET DEFAULT SYS$LOGIN
 (17:52)  $ SHOW DEFAULT
   USER$:[SMYTHE]
```

15.10.7 Example: Debugging Using the SHOW SYMBOL Command

The following example shows how the SHOW SYMBOL command is used to determine how the symbol COMMAND is defined:

```
$ SET VERIFY
$ @CLEAN
$ GET_COM_LOOP:
$    INQUIRE COMMAND -
     "ENTER COMMAND (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
  ENTER COMMAND (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): EXIT
$ SHOW SYMBOL COMMAND
 COMMAND = "EXIT"
$    IF COMMAND .EQS. "exit" THEN GOTO END_LOOP
   .
   .
   .
```

The SHOW SYMBOL command reveals that the symbol COMMAND has the value "EXIT". Because the INQUIRE command automatically converts input to uppercase and the IF statement that tests the command uses lowercase characters in the string "exit", DCL determines that the strings are not equal. To correct the error, make sure that the quoted string in the IF statement is written in capital letters. The rest of the string can use either uppercase or lowercase letters.

15.10.8 Enabling Verification During Execution

You can also interrupt a command procedure while it is executing to enable verification. As long as the command procedure does not contain the SET VERIFY command or a Ctrl/Y key sequence, you can enable verification by following these steps:

Step	Action
1	Press Ctrl/Y to interrupt execution.
2	Enter the SET VERIFY command.
3	Enter the CONTINUE command to continue execution of the command procedure (with verification enabled).

15.11 Step 6: Add Clean-Up Tasks

15.11.1 Cleaning Up

In general, execution of a command procedure should not change the user's process state. Therefore, a command procedure should include a set of commands that return the process to its original

state. This set of commands is usually part of a subroutine that is labeled "CLEAN_UP" (not to be confused with the previous example of CLEANUP.COM). Common clean up operations include closing files and resetting the default device and directory.

15.11.2 How to Add Clean Up Tasks

Follow this procedure to add clean up tasks to your command procedure:

Step	Task
1	Begin the cleanup subroutine with a label, such as CLEAN_UP.
2	Test for any open files using the F$GETJPI lexical function.
3	Delete any temporary or extraneous files using the DELETE or PURGE command.
4	If you have changed any defaults (such as the device or directory), restore them to their original state using the SET DEFAULT command.
5	Include an ON CONTROL_Y statement to ensure that the clean up operations are performed.

15.11.3 Closing Files

If you have any open files, make sure that they are closed before the procedure exits. You can use the lexical function F$GETJPI to examine the remaining open file quota (FILCNT) for the process. If FILCNT is the same at the beginning and end of the command procedure, you know that no files have been left open.

15.11.4 Example

These are the commands that you would use to warn a user that a file has been left open:

```
$ FIL_COUNT = F$GETJPI ("","FILCNT")
    .
    .
    .
$ IF FILCNT .NE. F$GETJPI ("", "FILCNT") THEN-
  WRITE SYS$OUTPUT "WARNING -- file left open)
```

15.11.5 Deleting Temporary or Extraneous Files

If you have created temporary files, delete them. In general, if you have updated any files, you should purge them to delete the previous copies. Before you delete files you have not created, make sure you want to delete them. For example, if you have updated a file that contains crucial data, you might want to make the purging operation optional.

15.11.6 Saving and Restoring Defaults

If you change the default device, the directory, or both, reset the original defaults before the command procedure exits. To save the name of the original default directory, use the DEFAULT keyword of the F$ENVIRONMENT lexical function. At the end of the command procedure, include a SET DEFAULT command that restores the saved device and directory.

15.11.7 Example

The command lines shown in this example save and restore the device and directory defaults:

```
$ SAV_DEFAULT = F$ENVIRONMENT ("DEFAULT")
    .
    .
    .
$ SET DEFAULT 'SAV_DEFAULT'
```

15.11.8 Commonly Changed Process Characteristics

The following table lists other commonly changed process characteristics, the lexical functions used to save them, and the lexical function or command used to restore them:

Characteristic	Lexical Function Used to Save	Lexical Function Used to Restore
DCL prompt	F$ENVIRONMENT	SET PROMPT
Default protection	F$ENVIRONMENT	SET PROTECTION/DEFAULT
Privileges	F$SETPRV	F$SETPRV or SET PROCESS /PRIVILEGES
Control characters	F$ENVIRONMENT	SET CONTROL
Verification	F$VERIFY	F$VERIFY
Message format	F$ENVIRONMENT	SET MESSAGE
Key state	F$ENVIRONMENT	SET KEY

For complete descriptions of these lexical functions, see the *OpenVMS DCL Dictionary*.

15.11.9 Ensuring Cleanup Operations Are Performed

To ensure that cleanup operations are performed even if the command procedure is aborted, begin each command level in the command procedure with the following statement:

```
$ ON CONTROL_Y THEN GOTO CLEANUP
```

For additional information on using the ON CONTROL_Y command, see Chapter 16.

15.12 Step 7: Complete the Command Procedure

15.12.1 How to Complete Command Procedures

When your general design works correctly, follow these steps to complete your command procedure:

Step	Task
1	Substitute commands for the first program stub in the command procedure.
2	Test the command procedure to make sure that the new commands work properly.
3	Debug the command procedure, if necessary.
4	When the first program stub works, move to the next one, and so on, until all program stubs have been replaced.

15.12.2 Example: Replacing a Program Stub with Commands

The following example shows the code for the TYPE section of CLEANUP.COM:

```
$! Execute if user entered TYPE
$! TYPE:
$       IF COMMAND .NES. "TYPE THEN GOTO ERROR
$       INQUIRE FILE "File to type"
$       TYPE 'FILE'
$       GOTO GET_COM_LOOP
```

This would replace the existing code:

```
$ WRITE SYS$OUTPUT "This is the TYPE section."
```

15.13 Example: CLEANUP.COM Command Procedure

Following is an example of the completed CLEANUP.COM command procedure:

```
$ GET_COM_LOOP:
$     INQUIRE COMMAND -
      "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
$     IF COMMAND .EQS. "EXIT" THEN GOTO END_LOOP
$!
$!Execute if user entered DELETE
$ DELETE:
$       IF COMMAND .NES. "DELETE" THEN GOTO DIRECTORY
$       INQUIRE FILE "File to delete? "
$       DELETE 'FILE'
$       GOTO GET_COM_LOOP
$!
$!Execute if user entered DIRECTORY
$ DIRECTORY:
$       IF COMMAND .NES. "DIRECTORY" THEN GOTO PRINT
$       DIRECTORY
$       GOTO GET_COM_LOOP
$!
```

```
$!Execute if user entered PRINT
$ PRINT:
$       IF COMMAND .NES. "PRINT" THEN GOTO PURGE
$       INQUIRE FILE "File to print? "
$       PRINT SYS$OUTPUT 'FILE'
$       GOTO GET_COM_LOOP
$!
$!Execute if user entered PURGE
$ PURGE:
$       IF COMMAND .NES. "PURGE" THEN GOTO TYPE
$       PURGE
$       GOTO GET_COM_LOOP
$!
$!Execute if user entered TYPE
$ TYPE:
$       IF COMMAND .NES. "TYPE" THEN GOTO ERROR
$       INQUIRE FILE "File to type"
$       TYPE 'FILE'
$       GOTO GET_COM_LOOP
$!
$ ERROR:
$       WRITE SYS$OUTPUT "You entered an invalid command."
$       GOTO GET_COM_LOOP
$!
$ END_LOOP:
$ WRITE SYS$OUTPUT "Directory ''F$DIRECTORY()' has been cleaned."
$
$ EXIT
```

Executing Command Procedures

15.14 Methods of Executing Command Procedures

15.14.1 Overview

To make a command procedure run, you must **execute** it. You can execute command procedures:

- From within another command procedure
- On remote nodes
- As parameters or qualifiers to DCL commands
- Interactively
- As batch jobs
- On disk and tape volumes

This section describes each of these methods.

15.15 Executing Command Procedures from Within Other Command Procedures

15.15.1 Command Line

You can execute another command procedure from within a command procedure by including an execute procedure command (@).

15.15.2 Example

The following command procedure, WRITEDATE.COM, invokes the command procedure GETDATE.COM:

```
$! WRITEDATE.COM
$!
$ INQUIRE TIME "What is the current time in hh:mm format?"
$ @GETDATE [JONES.COM]GETDATE.COM
```

15.16 Executing Command Procedures on Remote Nodes

15.16.1 Remote Nodes

You can use the TYPE command to execute command procedures in the top-level directory of another account on a remote node. You can execute command procedures that:

- Display the status of services in the local VMScluster that are not provided clusterwide
- List the users logged in to the remote node

15.16.2 Command Line

Enter the TYPE command followed by an access control string. Use the following format:

```
$ TYPE nodename"username password"::"TASK=command_procedure"
```

The variables username and password are the user name and password for the account on the remote node.

15.16.3 Example: SHOWUSERS Command Procedure

This command procedure displays the users logged in to the remote node on which the command procedure resides:

```
$!SHOWUSERS.COM
$!
$ IF F$MODE() .EQS. "NETWORK" THEN DEFINE/USER SYS$OUTPUT SYS$NET
$ SHOW USERS
```

15.16.4 Example: Executing SHOWUSERS

In the following example, SHOWUSERS.COM is located in the top-level directory of BIRD's account on node ORIOLE, and the password is BOULDER. SHOWUSERS.COM executes the DCL command SHOW USERS on the remote node ORIOLE. The TYPE command displays the output from SHOWUSERS.COM on the local node; that is, on the terminal from which you enter the type command:

```
$ TYPE ORIOLE"BIRD BOULDER"::"TASK=SHOWUSERS"

             OpenVMS User Processes at 11-DEC-1995 17:20:13.30
    Total number of users = 4, number of processes = 4

 Username     Node           Interactive  Subprocess   Batch
 FLICKER      AUTOMA                2         1
 ROBIN        FABLES                1         2         1
 DOVE         MURMUR                1
 DUCK         FABLES                1         1
```

15.16.5 Security Note

Your password will be visible on your terminal when you use the TYPE command with an access control string. Take the appropriate security precautions as described in Chapter 18.

15.17 Executing Command Procedures as Qualifiers or Parameters to DCL Commands

15.17.1 Overview

You can create a command procedure that specifies DCL command parameters or qualifiers. This type of command procedure is useful when there is a set of parameters or qualifiers that you use frequently with one or more commands.

15.17.2 Command Line

Enter the execute procedure command (@) in a command line where you would normally specify qualifiers or parameters.

15.17.3 Examples

- This command procedure can be used to enter a set of qualifiers to the LINK command:

```
$! This command procedure contains command
$! qualifiers for the LINK command.
$!
/DEBUG/SYMBOL_TABLE/MAP/FULL/CROSS_REFERENCE
```

  This command line links an object named SYNAPSE.OBJ, using the qualifiers specified in DEFLINK.COM:

```
$ LINK SYNAPSE@DEFLINK
```

- This command procedure can be used to enter the parameters CHAP1.TXT, CHAP2.TXT, and CHAP3.TXT with a DCL command:

```
$! PARAM.COM
$! This command procedure contains a list of
$! parameters that can be used with commands.
$!
CHAP1, CHAP2, CHAP3
```

  This command line specifies the command procedure PARAM in place of a list of parameters. In the following example, the parameters are the file names listed in PARAM.COM:

```
$ DIRECTORY/SIZE @PARAM
```

15.17.4 Restrictions

The following restrictions apply when executing command procedures:

- You cannot include a space before an execute procedure command (@) when the command procedure begins with a qualifier name.
- You must precede the execute procedure command (@) with a space when the command procedure begins with a parameter.

15.18 Executing Command Procedures Interactively

15.18.1 Command Line

Enter an execute procedure command (@) followed by the file specification of the command procedure.

15.18.2 Example

This command executes the procedure SETD.COM in the [MAINT.PROCEDURES] directory on the WORKDISK: disk:

```
$ @WORKDISK:[MAINT.PROCEDURES]SETD Return
```

15.18.3 Defining Symbols

You can define a symbol name to represent long command lines. You can then use the symbol to execute a command procedure.

15.18.4 Example

To use a symbol to execute the command procedure shown in the previous example, include this line in your login command procedure:

```
$ SETD == "@WORKDISK:[MAINT.PROCEDURES]SETD"
```

Then, to execute the procedure SETD.COM, enter the symbol name as you would any command:

```
$ SETD Return
```

15.18.5 Redirecting Interactive Output

By default, when you execute a command procedure interactively, the operating system displays output at your terminal. However, you can redirect output to a file by using the /OUTPUT qualifier to the execute command.

When you redirect command procedure output to a file, the procedure sends any error messages to the terminal and to the file that is receiving the output.

15.18.6 Example

This command writes the output from SETD.COM to the file RESULTS.TXT instead of to the terminal:

```
$ @SETD/OUTPUT=RESULTS.TXT
```

15.18.7 /OUTPUT Qualifier Restriction

Always place the /OUTPUT qualifier immediately after the command procedure name, with no intervening spaces. Otherwise, DCL interprets the qualifier as a parameter to be passed to the procedure.

15.19 Executing Command Procedures as Batch Jobs

15.19.1 Overview

If you use command procedures that require lengthy processing time (for example, compiling or assembling large programs), submitting these procedures as batch jobs will allow you to continue using your terminal interactively.

15.19.2 Submitting Batch Jobs

To execute a command procedure in batch mode, submit your command procedure to a batch queue (a list of batch jobs waiting to execute) by entering the DCL command SUBMIT. When you submit a job, it is directed to the default batch queue SYS$BATCH where it is added to the end of the queue of jobs waiting to be executed. When the jobs preceding yours are completed, your job is executed. On OpenVMS systems, the number of batch jobs that can execute simultaneously is specified when the batch queue is created by the system manager.

15.19.3 Example

The following example shows how to execute the command procedure named JOB1.COM. The SUBMIT command uses the default file type .COM; therefore you do not have to enter the file type if your command procedure has the file type .COM:

```
$ SUBMIT JOB1
Job JOB1 (queue SYS$BATCH, entry 651, started on SYS$BATCH))
```

15.19.4 Remote Batch Jobs

If your system is part of a network, you can submit a command procedure as a batch job on a remote node. Within a command procedure, you can use DCL commands to open and close files on remote notes and to read and write records in those files, using the same commands and qualifiers for local files.

15.19.5 Restarting Batch Jobs

By default, if the system fails before the job is finished, batch jobs are reexecuted beginning with the first line. However, you can use the following symbols in your command procedure to specify a different restarting point:

- $RESTART

  A global symbol whose value is true if the batch job has been started at least once before this execution. Do not specify a value for $RESTART; the system will assign the appropriate value.

- BATCH$RESTART

  A global symbol whose value you specify using the SET RESTART_VALUE command.

15.19.6 Using $RESTART and BATCH$RESTART

The following procedure describes how to use the $RESTART and the BATCH$RESTART symbols:

Step	Action
1	Begin each possible starting point of the procedure with a label.
2	As the first step in each section, equate the value of BATCH$RESTART to the label using the SET RESTART_ VALUE command.
3	At the beginning of the procedure, test $RESTART.
4	If $RESTART is true, issue a GOTO statement using BATCH$RESTART as the transfer label.

15.19.7 Example

This command procedure extracts a number of modules from a library, concatenates those modules, and then sorts the resulting file:

```
$! SORT_MODULES.COM
!
$! Set default to the directory containing
$! the library whose modules are to be sorted
$ SET DEFAULT WORKDISK:[ACCOUNTS.DATA83]
$!
$! Check for restarting
$ IF $RESTART THEN GOTO "BATCH$RESTART"
$!
$ EXTRACT_LIBRARIES:
$ SET RESTART_VALUE=EXTRACT_LIBRARIES
   .
   .
   .
$ CONCATENATE_LIBRARIES:
$ SET RESTART_VALUE=CONCATENATE_LIBRARIES
   .
   .
   .
$ SORT_FILE:
$ SET RESTART_VALUE=SORT_FILE
   .
   .
   .
$ EXIT
```

If this command procedure aborts, it reexecutes from the beginning of the file, from the statement labeled CONCATENATE_LIBRARIES, or from the statement labeled SORT_FILE, depending on the value of BATCH$RESTART. If you were extracting a number of separate modules, you could make each extraction a separate section.

15.20 Executing Command Procedures on Disk and Tape Volumes

15.20.1 Executing on Private Disk Volumes

When you submit a command procedure with the SUBMIT command, you cannot access files on allocated devices. You can, however, execute a command procedure that is located on a private disk volume if the volume is mounted with the /SHARE qualifier.

15.20.2 Executing on Tape Volumes

You can execute command procedures that reside on tape volumes if:

- The procedure does not invoke any other procedures
- The procedure does not issue any GOTO commands that refer to labels in the procedure preceding the GOTO command

If either of these conditions occur, you can execute the command procedure by doing the following:

Step	Action
1.	Copy the command procedure to a shared disk volume.
2.	Execute the command procedure on the shared disk volume.

Exiting, Interrupting and Error Handling Command Procedures

15.21 Exiting from Command Procedures

15.21.1 Definition: Command Level

When you use any of the methods described in this section to exit from a command procedure, you need to be aware of command levels.

A **command level** is an input stream for the DCL level interpreter. When you enter commands at your terminal, you are entering commands at command level 0. A simple interactive command procedure (such as CLEANUP.COM) executes at command level 1. When the procedure terminates and the DCL prompt reappears on your screen, you are back at command level 0.

15.21.2 Methods of Exiting

There are three ways to exit from a command procedure while it is executing:

- Place an EXIT command in the command procedure
- Place a STOP command in the command procedure
- Enter Ctrl/Y during the execution of the program

15.21.3 Exiting with the EXIT Command

If an exit is caused by the end of the procedure or an EXIT command, control returns to the next higher command level. You can return a status value to the next higher command level by specifying the value as the parameter of the EXIT command.

15.21.4 Example

If you invoke the command procedure called SUB at the DCL level and SUB calls the subroutine SUB1, the following occurs:

1. Exiting from SUB1 returns you to SUB at the command line following the call to SUB1.
2. Exiting from SUB returns you to DCL command level.

15.21.5 Exiting with the STOP Command

If an exit is caused by a STOP command, control always returns to DCL command level, regardless of the command level in which the STOP command executes.

If you execute the STOP command in a batch job, the batch job terminates.

15.21.6 Exiting with Ctrl/Y

You can interrupt a command procedure by pressing Ctrl/Y and then using the EXIT or STOP command to terminate the procedure. In this case, both the EXIT and STOP command return you to the DCL level.

15.21.7 Example

In the following example, the TESTALL procedure is interrupted by pressing Ctrl/Y. The EXIT command terminates processing of the procedure and returns you to DCL level. (Note that you can also enter the STOP command after you interrupt the procedure.)

```
$ @TESTALL Return
 Ctrl/Y
$ EXIT Return
$
```

15.21.8 Exit-Handling Routines

When you interrupt a command procedure, if the command (or image) that you interrupt declares any exit-handling routines, the EXIT command gives these routines control. However, the STOP command does not execute these routines.

15.22 Handling Error Conditions

15.22.1 Handling Errors

By default, the command interpreter executes an EXIT command when a command results in an error or severe error. This causes the procedure to exit to the previous command level. For other severity levels (success, warning, and informational), the command procedure continues.

When the system issues an EXIT command as part of an error-handling routine, it passes the value of $STATUS back to the previous command level, with one change. The command interpreter sets the high-order digit of $STATUS to 1 so that the command interpreter does not redisplay the message associated with the status value.

15.22.2 Handling Label Errors

There is one exception to the way that the command interpreter handles errors. If you reference a label in a command procedure and the label does not exist (for example, if you include the command GOTO ERR1 and ERR1 is not used as a label in the procedure), the GOTO command issues a warning and the command procedure exits.

15.22.3 Example: Displaying Errors

In the following example, the command procedure TEST.COM contains an error in the output file specification:

```
$ CREATE DUMMY.DAT\
THIS IS A TEST FILE
$ SHOW TIME
```

When you execute this procedure, the CREATE command returns an error in $STATUS and displays the corresponding message. The command interpreter then examines the value of $STATUS, determines that an error occurred, issues an EXIT command, and returns the value of $STATUS. When the procedure exits, the error message is not redisplayed because the CREATE command already displayed the message once. At DCL command level, you can see that $STATUS contains the error message but the high-order digit has been set to 1. For example:

```
$ @TEST
%CREATE-E-OPENOUT, error opening DUMMY.DAT as output
-RMS-F-SYN, file specification syntax error
%DCL-W-SKPDAT, image data (records not beginning with "$") ignored
$ SHOW SYMBOL $STATUS
  $STATUS = "%X109110A2"
$ WRITE SYS$OUTPUT F$MESSAGE(%X109110A2)
  %CREATE-E-OPENOUT, error opening !AS as output
```

15.22.4 Default Error Actions

The following table describes the default action taken when an error condition or a Ctrl/Y interruption occurs while a command procedure is executing. You can override these default actions with the ON, SET [NO]ON, and SET [NO]CONTROL=Y commands.

Interrupt	Default Action
Error or severe error	Procedure exits to the next command level.
Ctrl/Y at DCL command level or command level 1	Procedure is interrupted; the procedure can continue if no other image forces it to exit.
Ctrl/Y at command level lower than level 1	Procedure exits to the next higher command level.

15.23 Other Methods of Error Handling

15.23.1 ON Command

The ON command specifies an action to be performed if an error of a certain severity or greater severity occurs. If such an error occurs, the system takes the following actions:

- Performs the action specified by the ON command.
- Sets $STATUS and $SEVERITY to indicate the result of the specified ON action. In general, they are set to success.
- Resets the default error action (to exit if an error or severe error occurs).

An ON command action is executed only once. Therefore, after a command procedure performs the action specified in an ON command, the default error action is reset.

The action specified by an ON command applies only within the command level in which the command is executed. Therefore, if you execute an ON command in a procedure that invokes another procedure, the ON command action does not apply to the nested procedure.

15.23.2 ON Command Format

The format of the ON command is as follows:

ON condition THEN [$] command

Where "condition" is one of the following keywords:

ON Keyword	Action Taken
WARNING	Command procedure performs the specified action if a warning, error, or severe error occurs.
ERROR	Command procedure performs the specified action if an error or severe error occurs. The procedure continues if a warning occurs.
SEVERE_ERROR	Command procedure performs the specified action if a severe (fatal) error occurs. The procedure continues if a warning or error occurs.

If an ON command action is established for a specific severity level, the command interpreter performs the specified action when errors of the same or worse severity occur. When less severe errors occur, the command interpreter continues processing the file.

15.23.3 Example: Using the ON Command

This command can be used to override the default error handling so that a procedure exits when warnings, errors, or severe errors occur:

```
$ ON WARNING THEN EXIT
```

15.23.4 Example: Resuming After an Error

If your command procedure incudes this command, the command procedure executes normally until an error or severe error occurs:

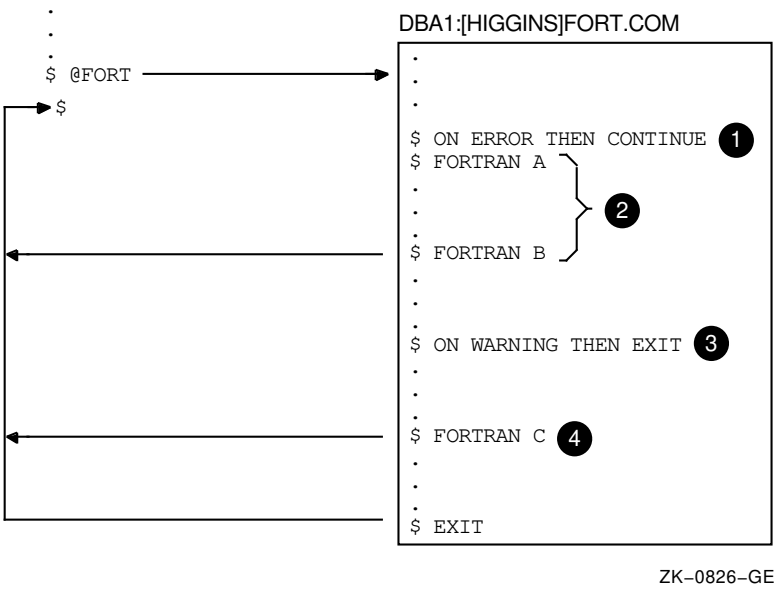
```
$ ON ERROR THEN GOTO ERR1
```

If such an error occurs, then the procedure resumes executing at ERR1. $STATUS and $SEVERITY are set to success and the default error action is reset. If a second error occurs before another ON or SET NOON command is executed, the procedure exits to the previous command level.

15.23.5 Figure: ON Command Actions

Figure 15–1 illustrates ON command actions.

Figure 15–1 ON Command Actions



ZK–0826–GE

❶ This ON command overrides the default command action (on warning, continue; on error or severe error, exit). If an error or severe error occurs while A.FOR is being compiled, the command procedure continues with the next command.

❷ The default command action is reset if the previous ON command takes effect. Thus, if an error or severe error occurs while both A.FOR and B.FOR are being compiled, the command procedure exits.

❸ If a warning, error, or severe error occurs while C.FOR is being compiled, the command procedure exits.

❹ If the command procedure does not exit before a command is executed, the command action takes effect.

15.23.6 For Additional Information

The sample command procedures FORTUSER.COM and CALC.COM in Appendix C also illustrate the use of the ON command to establish error handling.

15.24 Using the SET NOON Command

15.24.1 Disabling Error Checking

You can prevent the command interpreter from checking the status returned from commands by using the SET NOON command in your command procedure, which sets the ON command to NO status. When you use the SET NOON command, the command interpreter continues to place values in $STATUS and $SEVERITY but does not perform any error checking. You can restore error checking with the SET ON command or with an ON command.

When a procedure disables error checking, it can explicitly check the value of $STATUS following the execution of a command or program.

15.24.2 Example: Disabling Error Checking

In the following example, the SET NOON command preceding the RUN commands ensures that the command procedure continues if either of the programs TESTA or TESTB return an error condition. The SET ON command restores the default error checking by the command interpreter.

```
$ SET NOON
$ RUN TESTA
$ RUN TESTB
$ SET ON
```

15.24.3 Example: Checking the Value of $STATUS

In the following example, the first IF command checks whether $STATUS has a true value (that is, if it is an odd numeric value). If so, the FORTRAN command was successful and the LINK command executes. After the LINK command executes, $STATUS is tested again. If $STATUS is odd, the RUN command executes; otherwise, the RUN command does not execute. The SET ON command restores the current ON condition action; that is, whatever condition was in effect before the SET NOON command was executed:

```
$ SET NOON
$ FORTRAN MYFILE
$ IF $STATUS THEN LINK MYFILE
$ IF $STATUS THEN RUN MYFILE
$ SET ON
```

15.24.4 Command Levels

The SET ON or SET NOON command applies only at the current command level; that is, the command level at which the command is executed. If you use the SET NOON command in a command procedure that calls another command procedure, the default error-checking mechanism will be in effect within the nested procedure. Note that SET NOON has no meaning when entered interactively at DCL level.

15.25 Handling Ctrl/Y Interruptions

15.25.1 Ctrl/Y Interruptions

By default, when you press Ctrl/Y while a command procedure is executing, the command interpreter prompts for command input at a special command level called Ctrl/Y command level. From Ctrl/Y command level, you can enter DCL commands that are executed within the command interpreter and then resume execution of the command procedure with the CONTINUE command. In addition, you can stop the procedure by entering a DCL command that forces the command procedure to stop executing.

This section describes methods of overriding the way that command procedures process Ctrl/Y interruptions by using the ON command.

15.25.2 Stopping Command Procedures

You can interrupt a command procedure that is executing interactively by pressing Ctrl/Y. When you press Ctrl/Y, the command interpreter establishes a new command level, called the Ctrl/Y level, and prompts for command input. When the interruption occurs depends on the command or program that is executing:

- If the command is executed by the command interpreter itself (for example, IF, GOTO, or an assignment statement), the command completes execution before the command interpreter prompts for a command at the Ctrl/Y level.
- If the command or program is a separate image (that is, an image other than the command interpreter), the command is interrupted and the command interpreter prompts for a command at the Ctrl/Y level.

At the Ctrl/Y level, the command interpreter stores the status of all previously established command levels so that it can restore the correct status after any Ctrl/Y interrupt.

15.25.3 After Ctrl/Y Is Entered

After you interrupt a procedure, you can do the following:

- Enter a DCL command that is executed within the command interpreter.

  Among these commands are the SET VERIFY, SHOW TIME, SHOW TRANSLATION, ASSIGN, EXAMINE, DEPOSIT, SPAWN and ATTACH commands. After you enter one or more of these commands, you can resume the execution of the procedure with the CONTINUE command. See Section 16.10.5 for a complete list of commands that are executed within the command interpreter.

When you enter the CONTINUE command, the command procedure resumes execution with the interrupted command or program or with the line after the most recently completed command.

- Enter a DCL command that executes another image.

  When you enter any command that invokes a new image, the command interpreter returns to command level 0 and executes the command. This terminates the command procedure's execution. Any exit handlers declared by the interrupted image are allowed to execute before the new image is started.

- Enter the EXIT or STOP command to terminate the command procedure's execution.

  If you use the EXIT command, exit handlers declared by the interrupted image are allowed to execute. However, the STOP command does not execute these routines.

15.25.4 Note: Ctrl/Y on Exiting

If you do not exit from a command procedure (either explicitly from the command level or as part of an ON routine) following a Ctrl/Y, the next command you enter is interpreted in the context of the command procedure. For example, suppose you define the following symbol at the interactive level:

```
$ MAIL = "mail/edit=(send,reply,forward)"
```

If you enter Ctrl/Y to interrupt a command procedure that does not include this definition and then enter the command MAIL to send a message, your editor is not invoked automatically.

15.25.5 Stopping Privileged Images

If you interrupt the execution of a privileged image, you can enter only the CONTINUE, SPAWN, or ATTACH commands if you want to save the context of the image. If you enter any other commands (except from within a subprocess that you have spawned or attached to), the privileged image is forced to exit.

15.26 Setting Ctrl/Y Action Routines

15.26.1 Using the ON Command

The ON command, which defines an action to be taken in case of error conditions, also provides a way to define an action routine for a Ctrl/Y interruption that occurs during execution of a command procedure. The action that you specify overrides the default Ctrl/Y action (that is, to prompt for command input at the Ctrl/Y command level). For example:

```
$ ON CONTROL_Y THEN EXIT
```

If a procedure executes this ON command, a subsequent Ctrl/Y interruption during the execution of the procedure causes the procedure to exit. Control is passed to the previous command level.

15.26.2 When Ctrl/Y Is Pressed

When you press Ctrl/Y to interrupt a procedure that uses ON CONTROL_Y, the following actions are taken:

- If the command currently executing is a command executed within the command interpreter, the command completes and the Ctrl/Y action is taken.
- If the current command or program is executed by an image other than the command interpreter, the image is forced to exit and the Ctrl/Y action is taken. If the image has declared an exit handler, however, the exit handler is executed before the Ctrl/Y action is taken. The image cannot be continued following the Ctrl/Y action.

15.26.3 Effects of Entering Ctrl/Y

The execution of Ctrl/Y does not automatically reset the default Ctrl/Y action (that is, to prompt for command input at the Ctrl/Y command level). A Ctrl/Y action remains in effect until one of the following conditions occurs:

- The procedure terminates (as a result of pressing Ctrl/Y, executing an EXIT or STOP command, or a default error condition handling action).
- Another ON CONTROL_Y command is executed.
- The procedure executes the SET NOCONTROL=Y command (see Section 15.27).

A Ctrl/Y action can be specified in each active command level and affects only the command level in which it is specified.

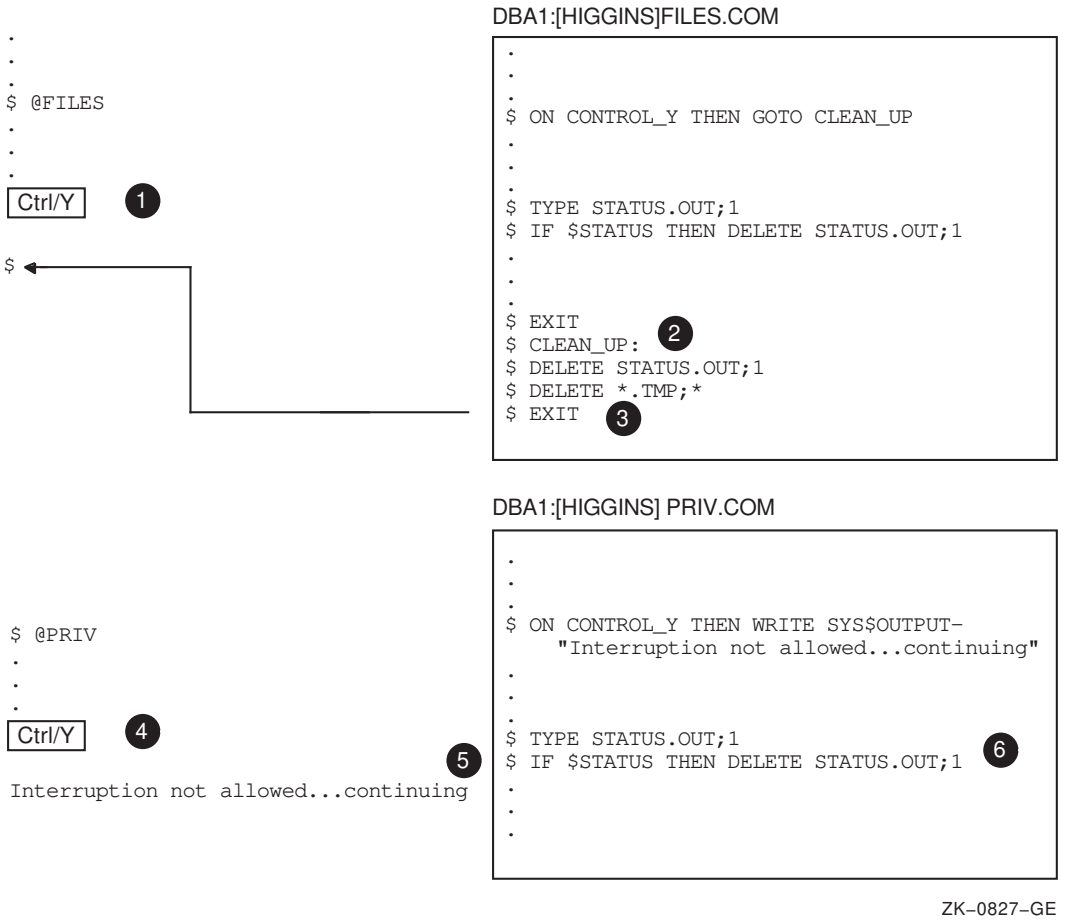
15.26.4 Example: Using the ON Command

When the command procedure shown in the following example executes, each Ctrl/Y interruption results in the execution of the SHOW TIME command. After each SHOW TIME command executes, the procedure resumes execution at the command following the command that was interrupted:

```
$ ON CONTROL_Y THEN SHOW TIME
```

15.26.5 Figure: Flow of Execution Following Ctrl/Y Action

The following figure illustrates the flow of execution following Ctrl/Y interruptions:



ZK–0827–GE

❶ The Ctrl/Y interruption occurs during the execution of the TYPE command.

❷ Control is then transferred to the label CLEAN_UP.

❸ After executing the routine, the command procedure exits and returns to the interactive command level.

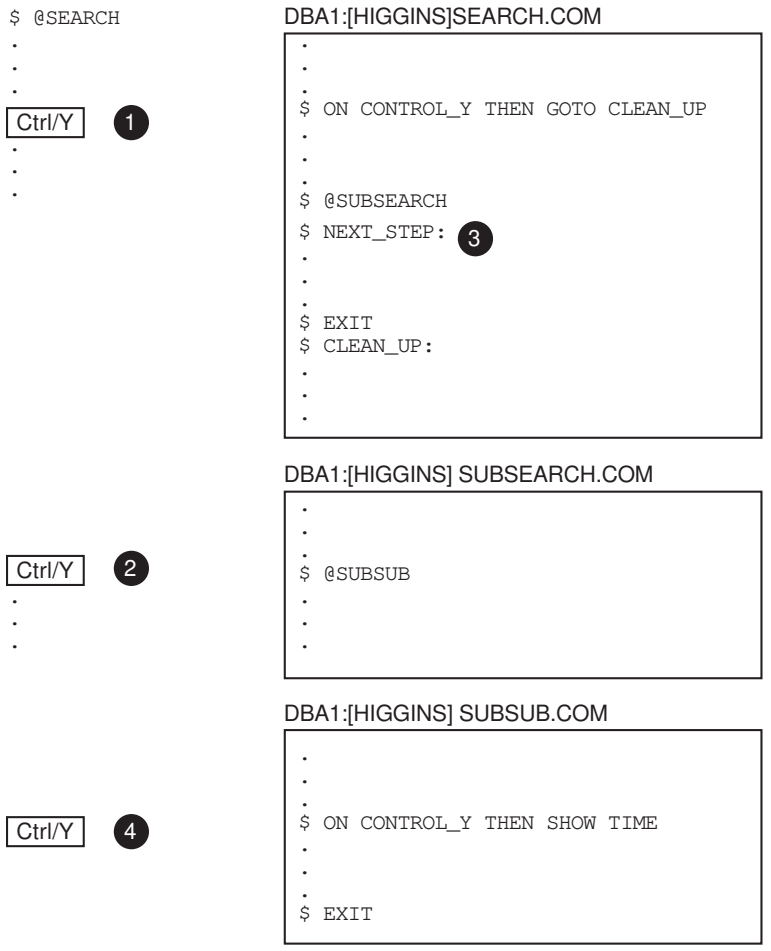
❹ The Ctrl/Y interruption occurs during the execution of the TYPE command.

❺ The WRITE command specified in the ON command is executed.

❻ The command procedure continues execution at the command following the interrupted command.

15.26.6 Figure: Ctrl/Y in Nested Procedures

The following figure illustrates what happens when Ctrl/Y is pressed during the execution of nested command procedures:



ZK–0828–GE

❶ If a Ctrl/Y interruption occurs while SEARCH.COM is executing, control is transferred to the label CLEAN_UP.

❷ If a Ctrl/Y interruption occurs while SUBSEARCH.COM is executing, control is transferred to the label NEXT_STEP in SEARCH.COM.

❸ Because no Ctrl/Y action is specified in SUBSEARCH.COM, the procedure exits to the previous command level when a Ctrl/Y interruption occurs.

❹ If a Ctrl/Y interruption occurs while SUBSUB.COM is executing, the SHOW TIME is executed.

15.27 Disabling and Enabling Ctrl/Y Interruptions

15.27.1 Using SET NOCONTROL=Y

The SET NOCONTROL=Y command disables Ctrl/Y handling. That is, if a command procedure executes the SET NOCONTROL=Y command, pressing Ctrl/Y has no effect.

The SET NOCONTROL=Y command also cancels the current Ctrl/Y action established with the ON CONTROL_Y command. To reestablish the default Ctrl/Y action, use the following two commands:

```
$ SET NOCONTROL=Y
$ SET CONTROL=Y
```

The SET NOCONTROL=Y command disables Ctrl/Y handling and cancels the current ON CONTROL_Y action. The SET CONTROL=Y command enables Ctrl/Y handling. At this point, the default action is reinstated. That is, if Ctrl/Y is pressed during the execution of the procedure, the command interpreter prompts for a command at the Ctrl/Y command level.

You can use the SET NOCONTROL=Y command at any command level. It affects all command levels until the SET CONTROL=Y command reenables Ctrl/Y handling.

15.27.2 Using the SET CONTROL=Y Command

An ON CONTROL_Y command remains in effect until another ON CONTROL_Y or a SET NOCONTROL=Y command executes or the command procedure exits.

To exit from a nonterminating loop when Ctrl/Y is disabled, you must delete your process from another terminal using the DCL command STOP. If you disable the default Ctrl/Y action, reset it as soon as possible. To reset the default Ctrl/Y action, execute the SET NOCONTROL=Y command followed by the SET CONTROL=Y command.

15.27.3 Disabling Ctrl/Y Interrupts

The ON CONTROL_Y and SET NOCONTROL=Y commands are intended for special applications. Digital does not recommend, in general, that you disable Ctrl/Y interruptions. To exit from a nonterminating loop when Ctrl/Y is disabled, you must delete (from another terminal) the process from which the looping procedure is executing.

15.27.4 Example: Using SET [NO]CONTROL=Y Commands

In this command procedure, pressing Ctrl/Y while a file is being typed passes control to the label END_TYPE:

```
   .
   .
   .
$! Type a file
$ IF COMMAND .NES. "TY" THEN GOTO END_TYPE
$ ON CONTROL_Y THEN GOTO END_TYPE
$ TYPE 'FILESPEC'
$END_TYPE:
$!
$! Reset default
$ SET NOCONTROL=Y
$ SET CONTROL=Y
   .
   .
   .
```

15.28 Detecting Errors in Command Procedures Using Condition Codes

15.28.1 Overview

When each DCL command in a command procedure completes execution, the command interpreter saves a condition code that describes the reason why the command terminated. This code can indicate successful completion or it can identify an informational or error message.

The command interpreter examines the condition code after it performs each command in a command procedure. If an error that requires special action has occurred, the system performs the action. Otherwise, the next command in the procedure executes.

15.28.2 Displaying Condition Codes ($STATUS)

The command interpreter saves the condition code as a 32-bit longword in the reserved global symbol $STATUS. The $STATUS symbol conforms to the format of a system message code as follows:

- Bits 0–2 contain the severity level of the message.
- Bits 3–15 contain the message number.
- Bits 16–27 contain the number associated with the facility that generated the message.
- Bits 28–31 contain internal control flags.

When a command completes successfully, $STATUS has an odd value. (Bits 0–2 contain a 1 or a 3.) When any type of warning or error occurs, $STATUS has an even value. (Bits 0–2 contain a 0, 2, or 4.) The command interpreter maintains and displays the current hexadecimal value of $STATUS. You can display the ASCII translation of $STATUS by entering the SHOW SYMBOL $STATUS command.

15.28.3 Example

In the following example, the file name (%FRED.LIS) is entered incorrectly:

```
$ CREATE %FILE.LIS
%CREATE-E-OPENOUT, error opening %FRED.LIS; as output
-RMS-F-WLD, invalid wildcard operation
$ SHOW SYMBOL $STATUS
  $STATUS = " %X109110A2"
$ WRITE SYS$OUTPUT F$MESSAGE(%X109110A2)
  %CREATE-E-OPENOUT, error opening !AS as output
```

15.28.4 Condition Codes with the EXIT Command

When a command procedure exits, the command interpreter returns the condition code for the previous command in $STATUS. The condition code provides information about whether the most recent command executed successfully.

When you use the EXIT command in a command procedure, you can specify a value that overrides the value that DCL would have assigned to $STATUS. This value, called a status code, must be specified as an integer expression.

When a command procedure contains nested procedures to create multiple command levels, you can use the EXIT command to return a value that explicitly overrides the default condition codes.

15.28.5 Example

Examine the following two command procedures:

```
$! This is file A.COM
$!
$ @B
   .
   .
   .


$! This is file B.COM
$!
$ ON WARNING THEN GOTO ERROR
   .
   .
   .
$ ERROR:
$ EXIT 1
```

The ON command in B.COM means that if any warnings, errors, or severe errors occur when B.COM is executing, the procedure is directed to the label ERROR. Here, the condition code is explicitly set to 1, indicating success. Therefore, when B.COM terminates, it passes a success code back to A.COM regardless of whether an error occurred.

15.28.6 Severity Levels

The low-order three bits of $STATUS represent the severity of the condition that caused the command to terminate. This portion of the condition code is contained in the reserved global symbol $SEVERITY. The $SEVERITY symbol can have the values 0 to 4, with each value representing one of the following severity levels:

Value	Severity
0	Warning
1	Success
2	Error
3	Information
4	Fatal (severe) error

Note that the success and information codes have odd numeric values, and warning and error codes have even numeric values.

15.28.7 Testing for Successful Completion

You can test for the successful completion of a command with IF commands that perform logical tests on $SEVERITY or $STATUS as follows:

```
$ IF $SEVERITY THEN GOTO OKAY
$ IF $STATUS THEN GOTO OKAY
```

These IF commands branch to the label OKAY if $SEVERITY and $STATUS have true (odd) values. When the current value in $SEVERITY and $STATUS is odd, the command or program completed successfully. If the command or program did not complete successfully, then $SEVERITY and $STATUS are even; therefore, the IF expression is false.

Instead of testing whether a condition is true, you can test whether it is false. For example:

```
$ IF .NOT. $STATUS THEN  . . .
```

The command interpreter uses the severity level of a condition code to determine whether to take the action defined by the ON command as described in Section 15.23.

15.29 Using Commands That Do Not Set $STATUS

Most DCL commands invoke system utilities that generate status values and error messages when they complete. However, there are several commands that do not change the values of $STATUS and $SEVERITY if they complete successfully. These commands are as follows:

CONTINUE	DECK	DEPOSIT
EOD	EXAMINE	GOTO
IF	RECALL	SET SYMBOL /SCOPE
SHOW STATUS	SHOW SYMBOL	STOP
WAIT		

If any of these commands result in a nonsuccessful status, the condition code is placed in $STATUS and the severity level is placed in $SEVERITY.

Login Command Procedures

15.30 Login Command Procedures

15.30.1 Overview

A login command procedure is a command procedure that the operating system automatically executes each time you log in. The system also executes this procedure at the beginning of every batch job that you submit.

There are 2 types of login command procedures:

- Systemwide (or group-defined)
- Personal

15.30.2 Systemwide Login Command Procedures

Systemwide login command procedures have the following characteristics:

- They are executed before your personal login command procedure
- When a systemwide login command procedure terminates, it passes control to your personal login command procedure
- They allow your system manager to make sure that certain commands are always executed when you log in

15.30.3 Creating Systemwide Login Command Procedures

To establish a systemwide login command procedure, your system manager equates the logical name SYS$SYLOGIN to the appropriate login command procedure. Your system manager can specify that this login command procedure be used for all system users or for a certain group of users.

15.30.4 Personal Login Command Procedures

You can create a personal login command procedure to execute the same commands each time you log in.

Your system manager assigns the file specification for your login command procedure. In most installations, the login command procedure is called LOGIN.COM. Therefore, you should name your login command procedure LOGIN.COM unless your system manager tells you otherwise.

15.30.5 Login Command Procedures in Captive Accounts

Your system manager can set up captive accounts by placing the name of a special command procedure in the LGICMD field for your account. If you log in to a captive account, you can perform only functions specified in the command procedure for your account; you cannot use the complete set of DCL commands. For more information on captive accounts, see the *OpenVMS System Manager's Manual*.

15.30.6 For More Information

Refer to Appendix C for a complete annotated LOGIN.COM procedure.

16

Complex Command Procedures: Programming with DCL

16.1 Overview

Complex command procedures can perform program-like functions. You can use variable input in a command procedure, execute sections of the procedure only if certain conditions are true, execute subroutines, or invoke other command procedures.

This chapter includes information about the following:

- Performing command procedure input
- Using parameters to pass data
- Using parameters to pass data to batch jobs
- Using parameters to pass data to nested command procedures
- Using the READ command to prompt for data
- Using the SYS$INPUT logical name to obtain data
- Performing command procedure output
- Writing data to terminals
- Redirecting output from commands and images
- Returning data from command procedures
- Redirecting error messages
- Reading and writing files (file I/O)
- Using the OPEN command
- Writing to files
- Using the WRITE command
- Using the READ command
- Using the CLOSE command
- Modifying files
- Creating output files
- Appending records to files
- Handling file I/O errors
- Using the IF command
- Using command blocks

- Using the GOSUB command
- Using the CALL command
- Writing Case statements

16.1.1 Who Should Read This Chapter

You should read this chapter if you:

- Have read Chapter 15
- Have basic knowledge of programming in DCL and would like to learn more advanced methods

16.1.2 For More Information

For additional information on the commands discussed in this chapter, refer to the *OpenVMS DCL Dictionary*.

16.2 Performing Command Procedure Input

16.2.1 Overview

Command procedures frequently require data provided by a user. This data, or input, can be obtained either interactively (as described in Chapter 15) or noninteractively. This chapter discusses noninteractive input methods, and different interactive methods than those described in Chapter 15.

16.2.2 Including Input Data

You can use the same data each time a command procedure executes. To do this, place the data in the command procedure on data lines following the command that requires the data.

16.2.3 Example

This command procedure executes the command procedure CENSUS.EXE. CENSUS.EXE reads the data 1991, 1992, and 1993 each time the procedure executes:

```
$ ! CENSUS.COM
$ !
$ RUN CENSUS
1992
1993
1994
$ EXIT
```

16.2.4 Restrictions to Including Data in Command Procedures

DCL passes the text on a data line directly to the command procedure. Therefore, it will not process data that must be translated such as:

- Symbols
- Logical names
- Arithmetic expressions

16.2.5 Other Methods of Inputting Data

Other methods of obtaining input data for command procedures that are described in the following sections are:

- Using parameters to pass data
- Using parameters to pass data to batch jobs
- Using parameters to pass data to nested command procedures
- Using the READ command to prompt for data
- Using the SYS$INPUT logical name to obtain data

16.3 Using Parameters to Pass Data

16.3.1 Rules for Passing Parameters

The following list contains rules and guidelines for passing parameters as data to command procedures:

- Place the parameters after the file specification of the command procedure.
- You can pass up to eight parameters to a command procedure.
- If you pass fewer than eight parameter values, the extra symbols are assigned null values. A null value is a string with no characters and is represented by quotation marks (" ").
- Separate the parameters with one or more spaces or tabs.

16.3.2 How Parameters Are Passed

DCL places parameters passed to command procedures in the local symbols P1 to P8. P1 is assigned to the first parameter value, P2 the second, P3 the third, and so on. For example, the following command invokes the command procedure SUM.COM and passes eight parameters to the procedure:

```
$ @SUM 34 52 664 89 2 72 87 3
```

16.3.3 Specifying Parameters as Integers

When you specify an integer as a parameter, it is converted to a string. In the following example, P1 is the string value 24; P2 is the string value 25:

```
$ @ADDER 24 25
```

You can use the symbols P1 to P8 in both integer and character string expressions; DCL performs the necessary conversions automatically.

16.3.4 Specifying Parameters as Character Strings

To preserve spaces, tabs, or lowercase characters in a character string, place quotation marks (" ") before and after the string. For example:

```
$ @DATA "Paul Cramer"
```

In the following example, P1 is Paul Cramer and P2 is null. If you omit the quotation marks, each character string is passed as a separate parameter. For example:

```
$ @DATA Paul Cramer
```

In this example, the strings Paul and Cramer are converted to uppercase letters; P1 is PAUL and P2 is CRAMER.

As another example, if you invoke DATA.COM with the following command:

```
$ @DATA  "Paul Cramer" 24 "(555) 111-1111")
```

P1 to P8 are defined in DATA.COM as follows:

P1 = Paul Cramer
P2 = 24
P3 = (555) 111-1111
P4–P8 = null

16.3.5 Specifying Parameters as Symbols

To pass the value of a symbol, place an apostrophe before and after the symbol. To preserve spaces, tabs, and lowercase characters in the symbol value, enclose the value in three sets of quotation marks. You must also use three sets of quotation marks to include a quotation mark as part of a string.

An alternative is to enclose the text in quotation marks and where a symbol appears, precede the symbol with two apostrophes and follow it with one apostrophe.

16.3.6 Examples

- In the following example, P1 is Paul and P2 is Cramer because DCL removes quotation marks when you pass a symbol to a command procedure:

  ```
  $ NAME = "Paul Cramer"
  $ @DATA 'NAME'
  ```

- In the following example, P1 is ‘‘Paul Cramer’’ and P2 is null:

  ```
  $ NEW_NAME = """Paul Cramer"""
  $ @DATA 'NEW_NAME'
  ```

- In the following example, P1 is translated to Paul Cramer:

  ```
  $ ! DATA.COM
  $ @NAME "''P1'"
  ```

16.3.7 Specifying Parameters as Null Values

To pass a null parameter, use one set of quotation marks as a placeholder in the command string. In the following example, the first parameter passed to DATA.COM is a null parameter:

```
$ @DATA "" "Paul Cramer"
```

In this example, P1 is null and P2 is Paul Cramer.

16.4 Using Parameters to Pass Data to Batch Jobs

16.4.1 The SUBMIT Command

To pass parameters to a command procedure executed in batch mode, use the SUBMIT command qualifier /PARAMETERS.

If you execute more than one command procedure using a single SUBMIT command, the specified parameters are used for each command procedure in the batch job.

16.4.2 Example

In the following example, the command passes three parameters to the command procedures ASK.COM and GO.COM, which are executed as batch jobs:

```
$ SUBMIT/PARAMETERS=(TODAY,TOMORROW,YESTERDAY) ASK.COM, GO.COM)
```

16.4.3 Example: Executing Multiple Command Procedures

In the following example, the SUBMIT command passes two parameters to the command procedures: LIBRARY.COM and SORT.COM:

```
$ SUBMIT-
_$ /PARAMETERS=(DISK:[ACCOUNT.BILLS]DATA.DAT,DISK:[ACCOUNT]NAME.DAT) -
_$ LIBRARY.COM, SORT.COM
```

The batch job executes as if you had logged in and executed each of the command procedures. This SUBMIT command executes a batch job that logs in under your account, executes your login command procedure, and then executes the following commands:

```
$ @LIBRARY DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT)
$ @SORT DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT)
```

16.4.4 Defining SYS$INPUT

You can also pass data to a batch job by including the data in a command procedure or by defining SYS$INPUT to be a file. The specified parameters are used for each command procedure in the batch job.

16.5 Using Parameters to Pass Data to Nested Command Procedures

16.5.1 Parameters in Nested Procedures

You can pass up to eight parameters to nested command procedures. The local symbols P1 to P8 in the nested procedure are not related to the local symbols P1 to P8 in the invoking procedure.

16.5.2 Example

In the following example, DATA.COM invokes the nested command procedure NAME.COM:

```
$ ! DATA.COM
$ @NAME 'P1' Joe Cooper
```

If P1 in DATA.COM is the string Paul Cramer, which contains no quotation marks, it is passed to NAME.COM as two parameters. In NAME.COM, P1 to P8 are defined as follows:

P1 = PAUL
P2 = CRAMER
P3 = JOE
P4 = COOPER
P5–P8 = null

If P1 in DATA.COM is "Paul Cramer' (quotation marks included), you can pass the value to NAME.COM as one parameter by enclosing P1 in three sets of quotation marks, as follows:

```
$ ! DATA.COM
$ QUOTE = """
$ P1 = QUOTE + P1 + QUOTE
$ @NAME 'P1' "Joe Cooper"
```

In this example, P1 is Paul Cramer and P2 is Joe Cooper in the command procedure NAME.COM.

16.6 Using the READ Command to Prompt for Data

16.6.1 Prompting for User Input

You can use the INQUIRE command (as described in Chapter 15) or the READ command to obtain data for command procedures interactively. Both commands prompt for input and assign the response to a symbol.

16.6.2 Comparing the INQUIRE and READ Commands

The READ command is different from the INQUIRE command in the following ways:

The INQUIRE command...	The READ command...
Prompts for a value	Prompts for a value
Reads the value from the terminal	Reads the value from the source specified by the first parameter
Assigns the value to a symbol	Assigns the value to the symbol named as the second parameter

16.6.3 The READ Command

The READ command accepts all characters typed on the terminal in response to the prompt, as an exact character string value (case, spaces, and tabs are preserved). If you omit the /PROMPT qualifier, the READ command displays Data: as the default prompt.

You can also write command procedures that can either accept parameters or prompt for user input if the required parameters are not specified.

16.6.4 Examples

- In the following example, the command issues the prompt Filename: to the terminal, reads the response from the source specified by the logical name SYS$COMMAND (by default, the terminal), and assigns the response to the symbol FILE:

```
$ READ/PROMPT="Filename: "  SYS$COMMAND  FILE
```

- In the following example, if a file name is not specified when the procedure is invoked, the user is prompted for a file name:

```
$ ! Prompt for a file name if name
$ ! is not passed as a parameter
$ IF P1 .EQS. "" THEN INQUIRE P1 "Filename"
$ COPY 'P1' DISK5:[RESERVED]*.*
$ EXIT
```

16.6.5 Note: Symbols and Batch Jobs

If you submit a command procedure for execution as a batch job, DCL reads the value for a symbol specified in an INQUIRE command from the data line following the INQUIRE command. If you do not include a data line, the symbol is assigned a null value.

16.7 Using the SYS$INPUT Logical Name to Obtain Data

16.7.1 Overview

Commands, utilities, and other system images get their input from the source specified by the logical name SYS$INPUT, which is the default input stream. In a command procedure, SYS$INPUT is defined as the command procedure file; commands or images that require data look for data lines in the file. However, by redefining SYS$INPUT, you can provide data from your terminal or from a separate input file.

16.7.2 Redefining SYS$INPUT as Your Terminal

You can redefine SYS$INPUT to be your terminal. This enables images called from command procedures to obtain input interactively, rather than from data lines in command procedures.

Note that you must redefine SYS$INPUT to be your terminal if you want to use a DCL command or utility that requires interactive input in command procedures.

16.7.3 Examples

- In the following example, the command procedure allows you to provide input interactively to the image CENSUS.EXE:

```
$ ! Execute CENSUS getting data from the terminal
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ RUN CENSUS
$ EXIT
```

The DEFINE/USER_MODE command temporarily redefines SYS$INPUT while CENSUS.EXE is running, so CENSUS.EXE obtains its input from the terminal. After

CENSUS.EXE completes, SYS$INPUT reverts to its original definition (the command procedure file).

- In the following example, the command procedure uses EVE as the text editor:

```
$ ! Obtain a list of your files
$ DIRECTORY
$ !
$ ! Get file name and invoke the EVE editor
$ EDIT_LOOP:
$       INQUIRE FILE "File to edit (Press Return to end)"
$       IF FILE .EQS. "" THEN EXIT
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$       EDIT/TPU 'FILE'
$       GOTO EDIT_LOOP
```

The command procedure prompts for file names until you terminate the loop by pressing the Return key. When you enter a file name, the procedure automatically invokes EVE to edit the file. While the editor is running, SYS$INPUT is defined as the terminal so you can enter your edits interactively.

16.7.4 Defining SYS$INPUT as a Separate File

A command procedure can also get input from a file by defining SYS$INPUT as a file. Note that DCL does not process data lines; command procedures pass text on data lines directly to commands or images. If you include DCL symbols or expressions on data lines, DCL will not substitute values for the symbols or evaluate the expressions. If you use an exclamation point (!) in a data line, the image to which you pass the data processes the exclamation point.

16.7.5 Including Programs in Data Files

You can also place programs in the command procedure file by specifying the name of the data file as SYS$INPUT. This causes the compiler to read the program from the command procedure rather than from another file.

16.7.6 Example

The following example shows a command procedure that contains a FORTRAN command followed by the program's statements:

```
$ FORTRAN/OBJECT=TESTER/LIST=TESTER SYS$INPUT
C  THIS IS A TEST PROGRAM
   A = 1
   B = 2
   STOP
   END
$ PRINT TESTER.LIS
$ EXIT
```

The FORTRAN command uses the logical name SYS$INPUT to identify the file to be compiled. Because SYS$INPUT equates to the command procedure, the FORTRAN compiler compiles the statements following the FORTRAN command (up to the next line that begins with a dollar sign). When the compilation completes,

two output files are created: TESTER.OBJ and TESTER.LIS. The PRINT command then prints the file.

16.8 Performing Command Procedure Output

16.8.1 Overview

Output from command procedures such as data, error messages, and verification of command lines can be directed to either terminals or other files. The following methods of directing output are covered in this section:

- Writing data to terminals
- Redirecting output from commands and images
- Returning data from command procedures
- Redirecting error messages

16.9 Writing Data to Terminals

16.9.1 Using the TYPE Command

Use the TYPE command to display text that is several lines long and does not require symbol substitution. The TYPE command writes data from the file you specify to SYS$OUTPUT.

16.9.2 Example

In the following example, SYS$INPUT is specified as the data file. The TYPE command reads data from the data lines that follow and displays the lines on the terminal:

```
$ ! Using TYPE to display lines
$ TYPE SYS$INPUT
REPORT BY MARY JONES
PREPARED APRIL 15, 1995
SUBJECT: Analysis of Tax Deductions for 1994
   .
   .
   .
$ EXIT
```

16.9.3 Using the WRITE Command

Use the WRITE command to write data that contains symbols or lexical functions. Unless you enclose the data in quotation marks (**" "**), the WRITE command performs symbol substitution automatically.

16.9.4 Character Strings as Literal Text

To use the WRITE command to display a character string as literal text, enclose the string in quotation marks (" "). For example:

```
$ WRITE SYS$OUTPUT "Two files are written."
Two files are written.
```

16.9.5 Quotation Marks in Character Strings

To include quotation marks in character strings, use two sets of quotation marks ("" ""). For example:

```
$ WRITE SYS$OUTPUT "Summary of ""Q & A"" Session"
Summary of "Q & A" Session
```

16.9.6 Character String Concatenation

To continue a line of text on more than one line, concatenate the two strings with a plus sign (+) and a hyphen (-). For example:

```
$ WRITE SYS$OUTPUT "Report by Mary Jones" + -
" Prepared April 15, 1995"
Report by Mary Jones Prepared April 15, 1995
```

16.9.7 Forcing Symbol Substitutions in Character Strings

The WRITE command performs symbol substitutions automatically and displays the values of symbols. To force symbol substitutions within character strings, enclose the symbol in apostrophes. For example:

```
$ AFILE = "STAT1.DAT"
$ BFILE = "STAT2.DAT"
$ WRITE SYS$OUTPUT "''AFILE' and ''BFILE' ready."
STAT1.DAT and STAT2.DAT ready.
```

In this example, STAT1.DAT is the translation of the symbol AFILE; STAT2.DAT is the translation of the symbol BFILE.

16.10 Redirecting Output from Commands and Images

16.10.1 Redirecting SYS$OUTPUT

Commands, utilities, and other system images write their output to the source specified by the logical name SYS$OUTPUT. By default, SYS$OUTPUT equates to the terminal. However, you can redirect the output in one of the following ways:

- Use the /OUTPUT qualifier when you invoke the command. DCL commands that accept the /OUTPUT qualifier include: ACCOUNTING, CALL, DIRECTORY, HELP, LIBRARY, RUN (process), SPAWN, and TYPE.
- Temporarily redefine SYS$OUTPUT as a file by using the DEFINE/USER_MODE command.
- Temporarily define SYS$OUTPUT as a null device (using the DEFINE/USER_MODE command) to suppress output from a command.

16.10.2 Example: Defining SYS$OUTPUT as a File

In the following example, the command procedure redirects the output from the SHOW USERS command to a file. The new definition for SYS$OUTPUT is in effect only for the execution of the SHOW USERS command:

```
$ DEFINE/USER_MODE SYS$OUTPUT SHOW_USER.DAT
$ SHOW USERS
$ !
$ ! Process the information in SHOW_USER.DAT
$ OPEN/READ INFILE SHOW_USER.DAT
$ READ INFILE RECORD
   .
   .
   .
$ CLOSE INFILE
$ EXIT
```

16.10.3 Example: Defining SYS$OUTPUT as a Null Device

In the following example, SYS$OUTPUT is defined as a null device (NL:).

```
$ DEFINE/USER_MODE SYS$OUTPUT NL:
$ APPEND NEW_DATA.DAT STATS.DAT
   .
   .
   .
```

16.10.4 Using the /USER_MODE Qualifier

The /USER_MODE qualifier is used to create a temporary logical name assignment that is in effect only until the next image completes. After the command executes, SYS$OUTPUT reverts to the default definition (usually the terminal).

You cannot use the DEFINE/USER_MODE command to redirect output from DCL commands that are executed within the command interpreter. Instead, use the DEFINE command to redefine SYS$OUTPUT and use the DEASSIGN command to delete the definition when you are through with it.

16.10.5 Commands Performed Within the Command Interpreter

The following is a complete list of DCL commands that are performed within the command interpreter:

=	ALLOCATE	ASSIGN
ATTACH	CALL	CANCEL
CLOSE	CONNECT	CONTINUE
CREATE/LOGICAL_ NAME_TABLE	DEALLOCATE	DEASSIGN
DEBUG	DECK	DEFINE
DEFINE/KEY	DELETE/SYMBOL	DISCONNECT
ELSE	ENDIF	ENDSUBROUTINE
EOD	EXAMINE	EXIT
GOSUB	GOTO	IF
INQUIRE	ON	OPEN
READ	RECALL	RETURN
SET CONTROL	SET DEFAULT	SET KEY

SET ON	SET OUTPUT_RATE	SET PROMPT
SET PROTECTION /DEFAULT	SET SYMBOL /SCOPE	SET UIC
SET VERIFY	SHOW DEFAULT	SHOW KEY
SHOW PROTECTION	SHOW QUOTA	SHOW STATUS
SHOW SYMBOL	SHOW TIME	SHOW TRANSLATION
SPAWN	STOP	SUBROUTINE
THEN	WAIT	WRITE

16.10.6 Example: Using the DEASSIGN Command

The following example shows the commands that would be used to redirect output from the SHOW TIME command to the file TIME.DAT. After you deassign SYS$OUTPUT, it reverts to the default definition (the terminal):

```
$ DEFINE SYS$OUTPUT TIME.DAT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
```

16.11 Returning Data from Command Procedures

16.11.1 Global Symbols and Logical Names

Global symbols and logical names return data from a command procedure to a calling procedure or to DCL command level. You can read a global symbol or a logical name at any command level. Logical names can return data from a nested command procedure to the calling procedure.

16.11.2 Example: Passing Values with Global Symbols

The following example shows a how a command procedure passes a value with a global symbol created with a global assignment statement:

```
$ @DATA "Paul Cramer"

$ ! DATA.COM
$ !
$ ! P1 is a full name.
$ ! NAME.COM returns the last name in the
$ ! global symbol LAST_NAME.
$ !
$ @NAME 'P1'
       $ ! NAME.COM
       $ ! P1 is a first name
       $ ! P2 is a last name
       $ ! return P2 in the global symbol LAST_NAME
       $ LAST_NAME == P2
       $ EXIT
$ ! write LAST_NAME to the terminal
$ WRITE SYS$OUTPUT "LAST_NAME = ''LAST_NAME'"

LAST_NAME = CRAMER
```

DATA.COM invokes the command procedure NAME.COM, passing NAME.COM a full name. NAME.COM places the last name in the global symbol LAST_NAME. When NAME.COM completes, DCL continues executing DATA.COM, which reads the last name by specifying the global symbol LAST_NAME. The command procedure NAME.COM would be in a separate file. It is shown indented in this example for clarity.

16.11.3 Example: Passing Values with Logical Names

In this command procedure, REPORT.COM obtains the file name for a report, equates the file name to the logical name REPORT_FILE, and executes a program that writes a report to REPORT_FILE:

```
$! Obtain the name of a file and then run
$! REPORT.EXE to write a report to the file
$!
$ INQUIRE FILE "Name of report file"
$ DEFINE/NOLOG REPORT_FILE 'FILE'
$ RUN REPORT
$ EXIT
```

16.11.4 Example: Logical Names with Calling Procedures

In the following example, the command procedure REPORT.COM is invoked from another procedure. The calling procedure uses the logical name REPORT_FILE to refer to the report file:

```
$! Command procedure that updates data files
$! and optionally prepares reports
$!
$ UPDATE:
   .
   .
   .
$   INQUIRE REPORT "Prepare a report [Y or N]"
$   IF REPORT THEN GOTO REPORT_SEC
$   EXIT
$!
$ REPORT_SEC:
$   @REPORT
$   WRITE SYS$OUTPUT "Report written to ", F$TRNLNM("REPORT_FILE")
$   EXIT
```

16.12 Redirecting Error Messages

16.12.1 Redefining SYS$ERROR

By default, command procedures send system error messages to the file indicated by SYS$ERROR. You can redefine SYS$ERROR to direct system error messages to a specified file. However, if you redefine SYS$ERROR to be different from SYS$OUTPUT (or if you redefine SYS$OUTPUT without also redefining SYS$ERROR), DCL commands and images that use standard system error display mechanisms send system error level and system severe level error messages to both SYS$ERROR and SYS$OUTPUT. Therefore, you receive these messages twice—once in the file indicated by the definition of SYS$ERROR and once in the file indicated by SYS$OUTPUT. Success, informational,

and warning level messages are sent only to the file indicated by SYS$OUTPUT.If you want to suppress system error messages from a DCL command, be sure that neither SYS$ERROR nor SYS$OUTPUT is equated to the terminal.

16.12.2 Error Messages

If you run one of your own images from a command procedure and the image references SYS$ERROR, the image sends system error messages only to the file indicated by SYS$ERROR—even if SYS$ERROR is different from SYS$OUTPUT. Only DCL commands and images that use standard system error display mechanisms send messages to both SYS$ERROR and SYS$OUTPUT when these files are different.

16.12.3 Example: Redefining SYS$ERROR

This command procedure accepts a directory name as a parameter, sets the default to that directory, and purges files in the directory. To suppress system error messages, the procedure temporarily defines SYS$ERROR and SYS$OUTPUT as the null device:

```
$ ! Purge files in a directory and suppress messages
$ !
$ SET DEFAULT 'P1'
$ ! Suppress messages
$ !
$ DEFINE/USER_MODE SYS$ERROR NL:
$ DEFINE/USER_MODE SYS$OUTPUT NL:
$ PURGE
$ EXIT
```

16.12.4 Suppressing System Error Messages

You can also use the SET MESSAGE command to suppress system messages. By using the qualifiers /NOFACILITY, /NOIDENTIFICATION, /NOSEVERITY, or /NOTEXT, you can suppress the facility name, message identification, severity level, or the message text.

16.12.5 Example

In the following example, the facility, identification, severity and text messages are temporarily suppressed, until the second SET MESSAGE command is issued:

```
$ ! Purge files in a directory and suppress system messages
$ !
$ SET DEFAULT 'P1'
$ ! Suppress system messages
$ !
$ SET MESSAGE/NOFACILITY -
             /NOIDENTIFICATION -
             /NOSEVERITY -
             /NOTEXT
$ PURGE
$ SET MESSAGE/FACILITY -
             /IDENTIFICATION -
             /SEVERITY
             /TEXT
$ EXIT
```

16.13 Reading and Writing Files (File I/O)

16.13.1 Reading and Writing Files

The basic steps in reading and writing files from command procedures are:

Step	Action
1	Use the OPEN command to open files. This assigns a logical name to the file and specifies whether the file is to be read, written, or both read and written. Subsequent READ, WRITE, and CLOSE commands use this logical name to refer to the file.
2	Use the READ or WRITE commands to read or write records to files. Input and output to files are usually accomplished by designing a loop to read a record, process the record, and write the modified record to either the same file or to another file.
3	Use the CLOSE command to close files. If you do not include the CLOSE command, files remain open until you log out.

16.13.2 Note: Process Permanent Files

You do not have to open process-permanent files such as SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, and SYS$ERROR explicitly to read or write to them because the system opens these files for you when you log in.

16.13.3 In This Section

The following sections describe:

- Using the OPEN command
- Writing to files
- Using the WRITE command
- Using the READ command

- Using the CLOSE command
- Modifying files
  - Updating records
  - Creating new output files
  - Appending records to files

16.14 Using the OPEN command

16.14.1 Overview

The OPEN command opens sequential, relative, or indexed sequential files. The files are opened as process-permanent; they remain open for the duration of your process unless you explicitly close them (with the CLOSE command). While the files are open, they are subject to OpenVMS RMS restrictions on using process-permanent files.

16.14.2 Opening Files

When you open a file, the OPEN command assigns a logical name (specified as the first parameter) to the file (specified as the second parameter) and places the name in the process logical name table. Subsequent READ, WRITE, and CLOSE commands use this logical name to refer to the file.

16.14.3 Example

In the following example, the OPEN command assigns the logical name INFILE to the file DISK4:[MURPHY]STATS.DAT:

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
```

16.14.4 Note: Logical Names

The logical name in the OPEN command must be unique. If the OPEN command does not work and your commands seem correct, change the logical name in the OPEN command. To display a list of logical name definitions, use the SHOW LOGICAL command.

16.14.5 Specifying Files

To ensure that the command procedure can access the correct files, use complete file specifications (for example, DISK4:[MURPHY]STATS.DAT) or use the SET DEFAULT command to specify the proper device and directory before you open a file.

You can also specify shareable files. The /SHARE qualifier enables other opened file. In addition, users can access shareable files with the DCL commands TYPE and SEARCH.

16.14.6 Reading Files

The OPEN/READ command opens the files, assigns logical names to the files, and places record pointers at the beginning of the files. When you open files for reading, you can read but not write records. Each time you read a record, the pointer moves to the next record.

16.14.7 Example

The OPEN/READ command in this command procedure opens the file STATS.DAT and assigns the logical name INFILE to the file:

```
$   OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
$ READ_FILE:
$   READ/END_OF_FILE=DONE INFILE DATA
$   GOTO READ_FILE
$ DONE:
$   CLOSE INFILE
$   EXIT
```

16.14.8 Writing Files

Use the OPEN/WRITE command when you want to write to a new file. The OPEN/WRITE command creates a sequential file in print file format. The record format for the file is variable with fixed control (VFC), with a 2-byte record header. The /WRITE qualifier cannot be used with the /APPEND qualifier.

If you specify a file that already exists, the OPEN/WRITE command opens a new file with a version number one greater than the existing file.

16.14.9 Example

The command procedure in the following example creates a new file (NAMES.DAT) that can be used for writing:

```
$   OPEN/WRITE OUTFILE DISK4:[MURPHY]NAMES.DAT
$ UPDATE:
$   INQUIRE NEW_RECORD "Enter name"
$   WRITE OUTFILE NEW_RECORD
$   IF NEW_RECORD .EQS. "" THEN GOTO EXIT_CODE
$   GOTO UPDATE
$ EXIT_CODE:
$   CLOSE OUTFILE
$   EXIT
```

16.14.10 Appending Records

The OPEN/APPEND command appends records to the end of an existing file. If you attempt to open a file that does not exist, an error occurs and the file is not opened. The /APPEND qualifier cannot be used with the /WRITE qualifier.

16.14.11 Example

In the following example, records are appended to the end of an existing file, NAMES.DAT:

```
$ OPEN/APPEND OUTFILE DISK4:[MURPHY]NAMES.DAT
$ INQUIRE NEW_RECORD "Enter name"
$ WRITE OUTFILE NEW_RECORD
   .
   .
   .
$ CLOSE OUTFILE
```

16.14.12 Reading and Writing Files

The OPEN/READ/WRITE command places the record pointer at the beginning of a file so you can read the first record. When you use this method to open a file, you can replace only the record you have read most recently; you cannot write new records to the end of the file. In addition, a revised record must be exactly the same size as the record being replaced.

16.14.13 Example

In the following example, the record pointer is placed at the beginning of the file STATS.DAT so the first record can be read:

```
$ OPEN/READ/WRITE FILE DISK4:[MURPHY]STATS.DAT
```

16.15 Writing to Files

16.15.1 How to Write to Files

To write to files, use the following procedure:

Step	Action
1	Open the file for writing.
2	Begin the write loop with a label. File I/O is always done in a loop unless you are writing or reading a single record.
3	Read the data to be written. Use the INQUIRE command or the READ command to read data into a symbol.
4	Test the data. Check the symbol containing the data. If the symbol is null (for example, if you press Return and enter no data on the line), you have reached the end of the data to be written to the file and you should go to the end of the loop. Otherwise, continue.
5	Write the data to the file. Use the WRITE command to write the value of the symbol (one record) to the file.
6	Return to the beginning of the loop. You remain within the loop until there is no more data to be written to the file.
7	End the loop and close the file.

16.15.2 Example

The following command procedure writes data to the new file STATS.DAT. If a file of that name exists, a new version is created:

```
$ ! Write a file
$ ON ERROR THEN EXIT                          ! Exit if the command
$ !                                           !   procedure cannot
$ !                                           !   open the file
$ OPEN/WRITE IN_FILE DISK4:[MURPHY]STATS.DAT ! Open the file
$ ON CONTROL_Y THEN GOTO END_WRITE            ! Close the file if you
$ !                                           !   quit execution with
$ !                                           !   Ctrl/Y
$ ON ERROR THEN GOTO END_WRITE                ! Close the file if an
$ !                                           !   error occurs
$WRITE:                                       ! Begin the loop
$ INQUIRE STUFF "Input data"                  ! Prompt for input
$ IF STUFF .EQS. "" THEN GOTO END_WRITE       ! Test for the end of
$ !                                           !   the file
$ WRITE IN_FILE STUFF                         ! Write to the file
$ GOTO WRITE                                  ! Go to the beginning
$END_WRITE:                                   ! End the loop
$ !                                           !
$ CLOSE IN_FILE                               ! Close the file
```

16.15.3 Creating Files with Unique File Names

To create a file with a unique name, use the F$SEARCH lexical function to see whether the name is already in the directory. (See the lexical function descriptions in the *OpenVMS DCL Dictionary* for more information about F$SEARCH.)

16.15.4 Example

This command procedure prompts the user for a file name, then uses the F$SEARCH lexical function to search the default directory for the name. If a file with that name already exists, control is passed to ERROR_1, the procedure prints the message ‘‘The file already exists’’ and control returns to the label GET_NAME. The procedure then prompts for another file name:

```
$ ! FILES.COM
$ !
$GET_NAME:
$ INQUIRE FILE "File"           ! Prompt the user for a file name
$ IF F$SEARCH (FILE) .NES. ""   ! Make sure the file name is unique
$ THEN
$    WRITE SYS$OUTPUT "The file already exists"
$    GOTO GET_NAME
$ ELSE
$    OPEN/WRITE IN_FILE 'FILE'  ! Open the file with WRITE access
$ ENDIF
   .
   .
   .
$ EXIT
```

16.16 Using the WRITE Command

16.16.1 Specifying Data

When you specify data for the WRITE command, follow the rules for character string expressions described in Chapter 14. You can specify data in the following ways:

- Specify data to be written as a character string expression. The WRITE command automatically substitutes symbols and lexical functions.
- Write a string to an output file as a literal character string. The WRITE command does not perform symbol substitution on strings enclosed in quotation marks.
- Combine literal strings with symbol names. To force symbol substitution, place the entire string within quotation marks and use double apostrophes before the symbol to identify it and a single apostrophe following it.

  Another way to combine literal strings with symbol names is to insert a comma before and after the symbol, place quotation marks around the delimited symbol, and enclose the entire character string in quotation marks. For example:

  ```
  $ WRITE OUTFILE "Count is ",COUNT,"."
  ```

- Use apostrophes in the WRITE command line to force symbol substitution.
- Combine literal strings and lexical functions by using apostrophes to force symbol substitution within character strings.

16.16.2 Example

```
$! Define symbols
$!
$ CREATED = "File created April 15, 1995"
$ COUNT = 4
$ P4 = "fourth parameter"
$!
$! Open the file DATA.OUT for writing
$!
$ OPEN/WRITE OUTFILE DISK4:[MURPHY]DATA.OUT
$!
$ WRITE OUTFILE CREATED                 ❶
$ WRITE OUTFILE "CREATED"               ❷
$!
$ WRITE OUTFILE "Count is ''COUNT'."    ❸
$ WRITE OUTFILE P'COUNT'                ❹
$!
$ WRITE OUTFILE "Mode is ''f$mode()'"   ❺
$!
$ CLOSE OUTFILE
```

```
$ TYPE DISK4:[MURPHY]DATA.OUT Return    ❻
File created April 15, 1995
CREATED
Count is 4.
fourth parameter
Mode is INTERACTIVE
$
```

As you examine the example, note the following:

❶ Specifies the data to be written as a character string expression.

❷ Writes the string *CREATED* to the output file as a literal character string.

❸ Combines literal strings with symbol names.

❹ Uses an apostrophe in the WRITE command line to force symbol substitution. In this example, the WRITE command substitutes a value for the symbol COUNT and performs symbol substitution on the resulting command string (P4).

❺ Combines literal strings and lexical functions.

❻ Displays the data written to the output file DATA.OUT by the preceding WRITE commands.

16.16.3 Using the /SYMBOL Qualifier

When the WRITE command writes a record, it positions the record pointer after the record just written. The WRITE command can write a record that is up to 2,048 bytes long.

Use the /SYMBOL qualifier to write a record if either of the following conditions exist:

- The record is longer than 1,024 bytes
- An expression in the WRITE command is longer than 255 bytes

See the description of the WRITE command in the *OpenVMS DCL Dictionary* for more information on writing long records.

16.16.4 Using the /UPDATE Qualifier

You can use the WRITE command with the /UPDATE qualifier to change a record rather than insert a new one. To use the /UPDATE qualifier, you must open the file for both reading and writing.

16.17 Using the Read Command

16.17.1 Reading from a File

Use the READ command to read a record and assign its contents to a symbol. You can use the READ command to read records that are less than or equal to 1,024 characters in length. To read data from a file, use the following procedure:

Step	Action
1	Open the file for reading.
2	Begin the read loop with a label. File I/O is always done in a loop unless you are reading or writing a single record.
3	Read the data from the file. Use the READ command with the /END_OF_FILE qualifier to read a record and assign its contents to a symbol. The /END_OF_FILE qualifier causes DCL to pass control to the label specified by the /END_OF_FILE qualifier when you reach the end of the file. Generally, you specify the label that marks the end of the read loop.
4	Process the data. When you read a file sequentially, process the current record before reading the next one.
5	Return to the beginning of the loop. You remain in the loop until you reach the end of the file.
6	End the loop and close the file.

16.17.2 Example

This command procedure reads and processes each record in the file STATS.DAT. The procedure executes the READ command repeatedly until the end-of-file status is returned. Then, the procedure branches to the line labeled END_READ:

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT  !Open the file
$ !
$READ_DATA:                                 !Begin the loop
$ READ/END_OF_FILE=END_READ INFILE RECORD   !Read a record; test for
$                                           !  end of file
$                                           ! Process the data
   .
   .
   .
$ GOTO READ_DATA                            !Go to the beginning
$                                           !   of the loop
$END_READ:                                  !End of loop
$ CLOSE INFILE                              !Close the file
$ EXIT
```

16.17.3 Specifying Symbols

When you specify a symbol name for the READ command, the command interpreter places the symbol name in the local symbol table for the current command level. If you use the same symbol name for more than one READ command, each READ command redefines the value of the symbol name. For example, in the preceding example, the READ command reads a new record from the input file (STATS.DAT) each time through the loop. It then uses this record to redefine the value of the symbol RECORD.

16.17.4 Using the /END_OF_FILE Qualifier

When you read from files, you generally read and process each record until you reach the end of the file. By using the /END_OF_FILE qualifier with the READ command, you can construct a loop to read records from a file, process the records, and exit from the loop when you have finished reading all the records.

Note that the labels you specify for /END_OF_FILE qualifiers are subject to the same rules as labels specified for a GOTO command. (See Chapter 15 for more information on using the GOTO command.)

16.17.5 Using the READ Command in Loops

You should always use the /END_OF_FILE qualifier when you use the READ command in a loop. Otherwise, when the error condition indicating the end-of-file is returned by the OpenVMS Record Management Services (OpenVMS RMS), the command interpreter performs the error action specified by the current ON command. For example, OpenVMS RMS returns the error status %RMS-E-EOF. This causes a command procedure to exit unless the procedure has established its own error handling.

16.17.6 Using the /INDEX and /KEY Qualifiers

To read records randomly from indexed sequential files, use the READ command qualifiers /INDEX and /KEY. These qualifiers specify that a record should be read from the file by finding the specified key in the index and returning the record associated with that key. If you do not specify an index, the primary index (0) is used.

After you read a record randomly, you can read the remainder of the file sequentially by using READ commands without the /KEY or /INDEX qualifiers.

16.17.7 Using the /DELETE Qualifier

You can use the READ command with the /DELETE qualifier to delete records from indexed sequential files. The /DELETE qualifier causes a record to be deleted from a file after it has been read. Use the /DELETE qualifier with the /INDEX and /KEY qualifiers to delete a record specified by a given key.

16.17.8 For Additional Information

For more information on the /DELETE, /INDEX, and /KEY qualifiers, see the description of the READ command in the *OpenVMS DCL Dictionary*.

16.18 Using the CLOSE Command

16.18.1 Closing Files

The CLOSE command closes a file and deassigns the logical name created by the OPEN command. Be sure to close all files you open in a command procedure before the command procedure terminates. If you fail to close an open file, the file remains open when the command procedure terminates and the logical name

assigned to the open file is not deleted from the process logical name table.

16.18.2 Example

In the following example, the CLOSE command closes the file STATS.DAT and deassigns the logical name INFILE:

```
$ OPEN INFILE DISK4:[MURPHY]STATS.DAT
    .
    .
    .
$ CLOSE INFILE
```

16.19 Modifying Files

16.19.1 In This Section

This section describes three methods of modifying files:

- Updating records
- Creating new output files
- Appending records

16.19.2 Updating records

When you use the updating method to modify records, you can make minor changes to a small number of records in a file. Because this method does not allow you to change the size of a record or the number of records in the file, use it only for files with formatted records (for example, in a data file).

To make minor changes in a file, use this procedure:

Step	Action
1	Open the file for both read and write access.
2	Use the READ command to read through the file until you reach the record that you want to modify.
3	Modify the record. In a sequential file, the text of this record must be exactly the same size as the original record. If the text of the modified record is shorter, pad the record with spaces, adding spaces to the end of the modified record until it is the same length as the original record. If the text of the modified record is longer, you need to create a new file.
4	Use the WRITE/UPDATE command to write the modified record back to the file.
5	Repeat steps 2 to 4 until you have changed all records you intend to change.
6	Use the CLOSE command to close the file. After you close the file, it contains the same version number as when you started, even though individual records have been changed.

16.19.3 Example

The following command procedure shows how to make changes to a sequential file by reading and updating individual records:

```
$! Open STATS.DAT and assign it the logical name FILE
$!
$ OPEN/READ/WRITE FILE DISK4:[MURPHY]STATS.DAT
$ BEGIN_LOOP:
$! Read the next record from FILE into the symbol RECORD
$     READ/END_OF_FILE=END_LOOP FILE RECORD
$! Display the record and see if the user wants to change it
$! If yes, get the new record.  If no, repeat loop
$!
$     PROMPT:
$          WRITE SYS$OUTPUT RECORD
$          INQUIRE/NOPUNCTUATION OK "Change? Y or N [Y] "
$          IF OK .EQS. "N" THEN GOTO BEGIN_LOOP
$          INQUIRE NEW_RECORD "New record"
$! Compare the old and new records
$! If old record is shorter than new record, issue an
$! error message. If old record and new record are the
$! same length, write the record. Otherwise pad the new
$! record with spaces so it is correct length
$!
$          OLD_LEN = F$LENGTH(RECORD)
$          NEW_LEN = F$LENGTH(NEW_RECORD)
$          IF OLD_LEN .LT. NEW_LEN THEN GOTO ERROR
$          IF OLD_LEN .EQ. NEW_LEN THEN GOTO WRITE_RECORD
$          SPACES = "              "
$          PAD = F$EXTRACT(0,OLD_LEN-NEW_LEN,SPACES)
$          NEW_RECORD = NEW_RECORD + PAD
$!
$     WRITE_RECORD:
$          WRITE/UPDATE FILE NEW_RECORD
$          GOTO BEGIN_LOOP
$!
$     ERROR:
$          WRITE SYS$OUTPUT "Error -- New record is too long"
$          GOTO PROMPT
$!
$     END_LOOP:
$          CLOSE FILE
$          EXIT
```

The system displays the record on the terminal and you are asked whether the record needs to be modified. If you choose to modify the record, a new record is read from the terminal and its length is compared to the length of the original record. If the original record is longer, extra spaces make the new record the same size. If the original record is shorter, the system displays an error message and you are again prompted for a new record.

16.20 Creating Output Files

16.20.1 Making Extensive Changes to Files

To make extensive changes to a file, open that file for read access and open a new file for write access. Because you are creating a new output file, you can modify the size of records, add records, delete records, or insert records.

The OPEN/WRITE command opens a new file for write access. The new file can have the same name as the original file and a version number one higher than the version number of the old file.

16.20.2 Note: Opening Files

To ensure that the correct file is opened for reading, you must open the existing file for read access before you open the new version for write access.

16.20.3 Creating Output Files

To create files that you can modify, use the following procedure:

Step	Action
1	Open the file for read access. This is the input file, the file you are modifying.
2	Open a new file for write access. This is the output file, the file that you are creating. If you give the output file the same name as the input file, the output file will have a version number one greater than the input file.
3	Use the READ command to read each record from the file you are modifying. As you read each record from the original file, decide how the record is to be treated. See Section 16.20.4 for information on how records are handled.
4	Continue reading and processing records until you have finished.
5	Use the CLOSE command to close both the input file and the output file.

16.20.4 Modifying Records

In the following table, the symbol RECORD contains the record read from the original file:

If...	Then...
There is no change to the record	Write the same symbol to the new file.
The record is changed	Use the INQUIRE command to read a different record into the symbol, then write the modified symbol to the new file.
The record is deleted	Do not write the symbol to the new file.
A record is inserted	Use a loop to read records into the symbol and to write the symbol to the new file.

16.20.5 Examples: Modifying Records

- In the following example, the symbol NEW_FILE is written to a new file:

```
$ ! No change
$ WRITE NEW_FILE RECORD
```

- In the following example, the INQUIRE commands is used write a modified symbol to a new file:

```
$ ! Change
$ INQUIRE NEW_RECORD "New record"
$ WRITE NEW_FILE NEW_RECORD
```

- In the following example, a loop is used to write the symbol to a new file:

```
$ ! Insertion
$LOOP:
$ !Get new records to insert
$ INQUIRE NEW_RECORD "New record"
$ IF RECORD .EQS. "" THEN GOTO END_LOOP
$ WRITE NEW_FILE NEW_RECORD
$ GOTO LOOP
$END_LOOP:
```

16.20.6 Example: Creating Output Files

The following example shows a command procedure that reads a record from an input file, processes the record, and copies the record into an output file:

```
$! Open STATS.DAT for reading and assign it
$! the logical name INFILE
$! Open a new version of STATS.DAT for writing
$! and assign it the logical name OUTFILE
$!
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
$ OPEN/WRITE OUTFILE DISK4:[MURPHY]STATS.DAT
$!
$ BEGIN_LOOP:
$! Read the next record from INFILE into the symbol RECORD
$!
$     READ/END_OF_FILE=END_LOOP INFILE RECORD
$! Display the record and see if the user wants to change it
$! If yes, get the new record
$! If no, write record directly to OUTFILE
$!
$     PROMPT:
$          WRITE SYS$OUTPUT RECORD
$          INQUIRE/NOPUNCTUATION OK "Change? Y or N [Y] "
$          IF OK .EQS. "N" THEN GOTO WRITE_RECORD
$          INQUIRE RECORD "New record"
$!
$     WRITE_RECORD:
$          WRITE OUTFILE RECORD
$          GOTO BEGIN_LOOP
$!
$! Close input and output files
$     END_LOOP:
$          CLOSE INFILE
$          CLOSE OUTFILE
$          EXIT
```

16.21 Appending Records to Files

16.21.1 Appending Records to Files

Use the following procedure to OPEN/APPEND command to append records to the end of existing files:

Step	Action
1	Use the OPEN command with the /APPEND qualifier to position the record pointer at the end of the file. The /APPEND qualifier does not create a new version of the file.
2	Use the WRITE command to write new data records.
3	Continue adding records until you are finished.
4	Use the CLOSE command to close the file.

16.21.2 Example

The following command procedure appends records to the end of the file named STATS.DAT:

```
$! Open STATS.DAT to append files and assign
$! it the logical name FILE
$!
$ OPEN/APPEND FILE DISK4:[MURPHY]STATS.DAT
$!
$ BEGIN_LOOP:
$! Obtain record to be appended and place this
$! record in the symbol RECORD
$!
$     PROMPT:
$          INQUIRE RECORD -
           "Enter new record (press RET to quit) "
$          IF RECORD .EQS. "" THEN GOTO END_LOOP
$! Write record to FILE
$!
$          WRITE FILE RECORD
$          GOTO BEGIN_LOOP
$!
$! Close FILE and exit
$!
$     END_LOOP:
$          CLOSE FILE
$          EXIT
```

16.22 Handling File I/O Errors

16.22.1 Using the /ERROR Qualifier

Use the /ERROR qualifier with the OPEN, READ, or WRITE command to suppress system error messages and to pass control to a specified label. If an error occurs during an input or output operation, the /ERROR qualifier overrides all other error-control mechanisms (except the /END_OF_FILE qualifier on the READ command).

16.22.2 Example

The following example uses the /ERROR qualifier with the OPEN command:

```
$ OPEN/READ/ERROR=CHECK FILE CONTINGEN.DOC
   .
   .
   .
$ CHECK:
$  WRITE SYS$OUTPUT "Error opening file"
```

The OPEN command requests that the file CONTINGEN.DOC be opened for reading. If the file cannot be opened (for example, if the file does not exist), the OPEN command returns an error condition and transfers control to the CHECK label.

16.22.3 Displaying F$STATUS

The error path specified by the /ERROR qualifier overrides the current ON condition established for the command level. If an error occurs and the target label is successfully given control, the reserved global symbol $STATUS retains the code for the error. You can use the F$MESSAGE lexical function in your error-handling routine to display the message in $STATUS.

16.22.4 Example

In the following example, the lexical function F$MESSAGE is used to display the contents of the F$STATUS lexical:

```
$ OPEN/READ/ERROR=CHECK FILE 'P1'
   .
   .
   .
$ CHECK:
$  ERR_MESSAGE = F$MESSAGE($STATUS)
$  WRITE SYS$OUTPUT "Error opening file: ",P1
$  WRITE SYS$OUTPUT ERR_MESSAGE
   .
   .
   .
```

16.22.5 Default Error Actions

If an error occurs while you are using the OPEN, READ, WRITE, or CLOSE command and you do not specify an error action, the current ON command action is taken.

When a READ command receives an end-of-file message, the error action is determined as follows:

- If you use the /END_OF_FILE qualifier, control is passed to the specified label.
- If you do not use the /END_OF_FILE qualifier, control is passed to the label specified with the /ERROR qualifier.
- If you do not specify either qualifier (/END_OF_FILE or /ERROR), the current ON command action is taken.

Techniques for Controlling Execution Flow

16.22.6 Overview

The normal flow of execution in a command procedure is sequential: the commands in the procedure execute in order until the end of the file is reached. However, you can also control whether certain statements are executed or the conditions under which the procedure should continue executing.

16.22.7 In This Section

This section discusses:

- The DCL commands that you can use to control or alter the flow of execution:
  - IF
  - GOTO
  - GOSUB
  - CALL
- Using command blocks
- Writing case statements
- Writing loops

16.23 Using the IF Command

16.23.1 IF Command

The IF command tests the value of an expression and executes a command or block of commands when the result of the expression is true. When the result of the expression is false, one of the following occurs:

- When one command follows the THEN command, it is not executed and the following command executes.
- When a block of commands follows the THEN command and the ELSE command is not specified, the command immediately following the ENDIF command executes.
- When the ELSE command is specified, the command or block of commands following the ELSE command executes.

16.23.2 Formats for the IF Command

DCL provides two distinct formats for the IF command. The first format executes a single command when the expression specified to the IF command is true, as discussed in Chapter 15

DCL also provides a block-structured IF format. The block-structured IF command executes more than one command if the expression specified is true and accepts an optional ELSE statement, which executes one or more commands if the expression is false.

16.23.3 Using the THEN Command

To execute more than one command when an expression is true, specify the THEN command as a verb (a DCL command preceded by a dollar sign) and terminate the resulting block-structured statement with an ENDIF statement.

16.23.4 Example

In the following example, the THEN statement is used as a verb:

```
$ IF expression
$   THEN
$       command
$       command
    .
    .
    .
$ ENDIF
```

16.23.5 Using the ELSE Command

To execute one or more commands when an expression is false, specify the ELSE statement as a verb and terminate the resulting block-structured statement with an ENDIF statement.

16.23.6 Example

In the following example, the ELSE command is used as a verb:

```
$ IF expression
$   THEN
$       command
$       command
    .
    .
    .
$   ELSE
$       command
$       command
    .
    .
    .
$ ENDIF
```

16.24 Using Command Blocks

16.24.1 Executing Command Blocks

Command blocks can be executed in several ways, depending on whether you leave the commands in the same command procedure or put them in another command procedure and execute them there. The guidelines are as follows:

- If you leave the commands in the command procedure, place them after the THEN statement. For example:

```
$ IF condition
$ THEN  command
        command
   .
   .
   .
$ ENDIF
```

- If you place the commands in a separate procedure, make the call to that command procedure as part of the THEN statement. For example:

```
$ IF condition
$ THEN @command_procedure
$ ELSE command
$      command
$ ENDIF
```

- You can continue to specify the nonblock-structured IF format and direct flow to a labeled region when the condition specified is met. For example:

```
$ IF not condition THEN GOTO END_LABEL
   .
   .
   .
$END_LABEL:
```

16.24.2 Command Blocks and THEN Commands

You can execute a block of commands after the THEN command when the result of the IF expression is true. When you use a block of commands, place the THEN command as the first command on the line following the IF command.

16.24.3 Example: Command Blocks and THEN Commands

In the following example, two SET TERMINAL commands execute and the procedure transfers control to the label PROCEED when F$MODE equals ‘‘INTERACTIVE’’. When F$MODE does not equal ‘‘INTERACTIVE’’, the procedure exits:

```
$ IF F$MODE () .EQS. "INTERACTIVE"
$ THEN
$    SET TERMINAL/DEVICE=VT200
$    SET TERMINAL/WIDTH=132
$    GOTO PROCEED
$ ENDIF
$ EXIT
$PROCEED:
```

16.24.4 Example: Command Blocks And Else Commands

The following example illustrates how to use a block of commands with the IF command in conjunction with the ELSE command:

```
$ INQUIRE DEV "Device to check"
$ IF F$GETDVI(DEV, "EXISTS")
$ THEN
$    WRITE SYS$OUTPUT "The device exists."
$    SHOW DEVICE 'DEV'
$    SET DEVICE/ERROR_LOGGING 'DEV'
$ ELSE
$    WRITE SYS$OUTPUT "The device does not exist."
$    WRITE SYS$OUTPUT "Error logging has not been enabled."
$ ENDIF
$ EXIT
```

When the condition is true, the procedure writes a message to SYS$OUTPUT and executes the SHOW DEVICE and SET DEVICE commands. When the condition is not true, the procedure writes two messages to SYS$OUTPUT.

16.24.5 IF Command Restrictions

When you use the IF-THEN-ELSE language construct, observe the following restrictions:

- Include no more than 15 levels of nested IF statements.
- Terminate a command block started by a THEN statement with either an ELSE or an ENDIF statement.
- Terminate a command block started by an ELSE statement with an ENDIF statement.
- Include a THEN statement as the first executable statement following an IF statement.
- Do not specify a label on a line containing a THEN or an ELSE statement. You can, however, specify a label on a line containing an ENDIF statement. Programs can branch within the current command block but branching into the middle of another command block is not recommended.

16.24.6 True Expressions

The expression following the IF command can consist of one or more numeric constants, string literals, symbolic names, or lexical functions separated by logical, arithmetic, or string operators. An expression is true when it has one of the following values:

- An odd integer value
- A character string value that begins with any of the letters Y, y, T, or t
- A character string value that contains numbers that form an integer with an odd value (for example, the string "27")

16.24.7 False Expressions

An expression is false when it has one of the following values:

- An even integer value
- A character string value that begins with any letter other than Y, y, T, or t
- A character string value that contains numbers that form an integer with an even value (for example, the string "28")

16.24.8 Writing Expressions

When you write an expression for an IF command, adhere to the following rules:

- When you use symbols in IF statements, their values are automatically substituted. Do not use apostrophes (') as substitution operators unless you need to force iterative translation.
- String comparison operators end in the letter S. For example, use operators such as .EQS., .LTS., and .GTS. to compare strings. By contrast, the operators .EQ., .LT., and .GT. are used for comparing integers.
- When you test to see whether two strings are equal, the strings must use the same case in order for DCL to find a match. That is, the string ''COPY'' does not equal the string ''copy'' or the string ''CoPy.''

The following examples illustrate expressions that can be used with the IF command. For additional examples, see the description of the IF command in the *OpenVMS DCL Dictionary*.

16.24.9 Example: Using Logical Operators

The following example uses a logical operator and executes only one command following the THEN statement. When the symbol CONT is not true, the procedure exits:

```
$ INQUIRE CONT "Do you want to continue [Y/N]"
$ IF .NOT. CONT THEN EXIT
   .
   .
   .
```

16.24.10 Using Symbols

The following example uses a symbol and a label within the IF expression:

```
$ INQUIRE CHANGE "Do you want to change the record [Y/N]"
$ IF CHANGE THEN GOTO GET_CHANGE
   .
   .
   .
$ GET_CHANGE:
   .
   .
   .
```

When the symbol CHANGE is true, the procedure transfers control to the label GET_CHANGE. Otherwise, the procedure executes the command following the IF command.

16.24.11 Example: Comparing Integers and Strings

The following example illustrates two different IF commands:

```
$ COUNT = 0
$ LOOP:
$    COUNT = COUNT + 1
$    IF COUNT .EQ. 9 THEN EXIT
$    IF P'COUNT' .EQS. "" THEN EXIT
   .
   .
   .
$ GOTO LOOP
```

The first IF command compares two integers; the second IF command compares two strings. Note that the .EQ. operator is used for the integer comparison and the .EQS. operator is used for the string comparison.

First, the value of COUNT is compared to the integer 9. When the values are equal, the procedure exits; when the values are not equal, the procedure continues. The loop terminates after eight parameters (the maximum number allowed) have been processed.

In the second IF expression, the string value of the symbol P'COUNT' is compared to a null string to see whether the symbol is undefined. Note that you must use apostrophes to force iterative substitution of the symbol COUNT. For example, when COUNT is 2, the result of the first translation is P2. Then, the value of P2 is used in the string comparison.

16.24.12 Example: Executing Other Command Procedures

The following example executes the command procedure EXIT_ROUTINE.COM when the result of the IF expression is true:

```
$ GET_COMMAND_LOOP:
$    INQUIRE COMMAND -
     "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
$    IF COMMAND .EQS. "EXIT" THEN @EXIT_ROUTINE
```

16.24.13 GOTO Command

The GOTO command passes control to a labeled line in a command procedure. (For additional information on label usage, refer to Chapter 15.) The GOTO command is especially useful within a THEN clause to cause a procedure to branch forward or backward. For example, when you use parameters in a command procedure, you can test the parameters at the beginning of the procedure and branch to the appropriate label.

The target label of a GOTO or GOSUB command cannot be inside either a separate IF-THEN-ELSE construct or a separate subroutine.

16.24.14 Example: Using the GOTO Command

In the following example, the IF command checks that P1 is not a null string:

```
$ IF P1 .NES. "" THEN GOTO OKAY
$ INQUIRE P1 "Enter file spec"
$ OKAY:
$ PRINT/COPIES=10 'P1'
   .
   .
   .
```

If P1 is a null string, the GOTO command is not executed and the INQUIRE command prompts for a parameter value. Otherwise, the GOTO command causes a branch around the INQUIRE command. In either case, the procedure executes the PRINT command following the line labeled OKAY.

16.24.15 Example: Incorrectly Placed GOTO Command

In the following example the GOTO command returns an error message because its target (TEST_1) is within an IF-THEN construct:

```
$ GOTO TEST_1
$ EXIT
$ IF 1.EQ.1
$       THEN WRITE SYS$OUTPUT "What are we doing here?"
$ TEST_1:
$       WRITE SYS$OUTPUT "Got to the label"
$ ENDIF
$ EXIT
```

16.24.16 Avoiding Reexecution

You can also use the GOTO command to avoid reexecuting parts of the job that have completed successfully. To do this, follow these steps:

Step	Action
1	Begin each possible starting point in the procedure with a label.
2	After the label, use the SET RESTART_VALUE = label-name command to set the restarting point to that label. If the batch job is interrupted after the SET RESTART_VALUE = label-name command executes, the system assigns the appropriate label name to the global symbol BATCH$RESTART when the batch job restarts.
3	At the beginning of the procedure, test the value of the symbol $RESTART. If $RESTART is true, execute a GOTO statement using the symbol BATCH$RESTART as the transfer label.

16.24.17 The $RESTART Global Symbol

$RESTART is a reserved global symbol that the system maintains for you. $RESTART is true if one of your batch jobs was restarted after it was interrupted. Otherwise, $RESTART is false. You cannot delete the reserved global symbol $RESTART.

If a command procedure has SET RESTART_VALUE commands in it but you want the job to reexecute in its entirety, enter the SET ENTRY/NOCHECKPOINT command to delete the global symbol BATCH$RESTART. If you restart a job that was interrupted, the job starts executing in the section where it was interrupted.

16.24.18 Example

This command procedure shows how to use restart values in a batch job:

```
$ ! Set default to the directory containing
$ ! the file to be updated and sorted
$ SET DEFAULT DISK1:[ACCOUNTS.DATA84]
$
$ ! Check for restarting
$ IF $RESTART THEN GOTO 'BATCH$RESTART'
$
$ UPDATE_FILE:
$ SET RESTART_VALUE = UPDATE_FILE
   .
   .
   .
$ SORT_FILE:
$ SET RESTART_VALUE = SORT_FILE
   .
   .
   .
EXIT
```

To submit this command procedure as a batch job that can be restarted, use the /RESTART qualifier to the SUBMIT command when you submit the job. Because interrupted jobs begin executing in the section where they are interrupted, if this job is interrupted during the SORT_FILE routine, it starts executing at the label SORT_FILE when it is restarted.

16.24.19 Symbols in System Failures

Most of your process environment is not maintained when the system fails. The only symbols maintained across a system failure are $RESTART and BATCH$RESTART. Therefore, you should redefine any symbols or process logical names used in your command procedure after each SET RESTART_VALUE command or in a THEN block that executes if $RESTART is true. If you define symbols and logical names in a THEN block, the command GOTO 'BATCH$RESTART' should be the last command in the THEN block.

16.25 Using the GOSUB Command

16.25.1 GOSUB Command

The GOSUB command transfers control to a labeled subroutine in a command procedure. If the label does not exist in the command procedure, the procedure cannot continue executing and is forced to exit. (For complete information on labels, refer to Chapter 15.)

You can nest the GOSUB command up to 16 times per procedure level.

The GOSUB command is a local subroutine call; it does not create a new procedure level. Consequently, all labels and local symbols defined in the current command level are available to a subroutine invoked with GOSUB.

16.25.2 RETURN Command

The RETURN command terminates a subroutine and returns control to the command following the GOSUB command. You can specify a value for $STATUS with the RETURN command that overrides the value that DCL assigns to $STATUS at the end of the subroutine. This value must be an integer between zero and four or an equivalent expression. If you specify a value for $STATUS, DCL interprets this value as a condition code. If you do not specify a value for $STATUS, the current value of $STATUS is saved.

16.25.3 Example: Using the GOSUB Command

The following example shows how you can use the GOSUB command to transfer control to subroutines:

```
$!
$! GOSUB.COM
$!
$ SHOW TIME
$ GOSUB TEST1            ❶
$ WRITE SYS$OUTPUT "GOSUB level 1 has completed successfully."
$ SHOW TIME
$ EXIT
$!
$! TEST1 GOSUB definition
$!
$ TEST1:
$     WRITE SYS$OUTPUT "This is GOSUB level 1."
$     GOSUB TEST2        ❷
$     RETURN %X1         ❸
$!
$! TEST2 GOSUB definition
$!
$ TEST2:
$     WRITE SYS$OUTPUT "This is GOSUB level 2."
$     WAIT 00:00:02
$     RETURN             ❹
```

As you examine the example, note the following:

❶ The first GOSUB command transfers control to the subroutine labeled TEST1.

❷ The procedure executes the commands in subroutine TEST1, branching to the subroutine labeled TEST2.

❸ The RETURN command in subroutine TEST1 returns control to the main command procedure and passes a value of 1 to $STATUS, indicating successful completion.

❹ The RETURN command in subroutine TEST2 returns control to subroutine TEST1. Note that this command executes before command 3.

16.26 Using the CALL Command

16.26.1 Command Levels

There are two ways to create new command levels:

- Nest command procedures by using an execute procedure (@) command inside one command procedure to invoke another command procedure (as described in Chapter 15.)
- Use the CALL command to call a subroutine that exists within the command procedure.

16.26.2 CALL Command

The CALL command transfers control to a labeled subroutine in a command procedure and creates a new procedure level. The CALL command allows you to keep more than one related command procedure in a single file, making the procedures easier to manage. The subroutine label, which must be unique, can precede or follow the CALL command in the command procedure. Chapter 15 contains rules for entering subroutine labels.

In addition to the label, you can pass up to eight optional parameters to the subroutine. For complete information on parameters, refer to Section 16.3.

16.26.3 CALL Command Rules

Following are rules for using the CALL command:

- Sends output to SYS$OUTPUT
- Has an optional /OUTPUT qualifier that allows you to direct output from the subroutine to a file
- Uses a default file type .LIS for the output file
- Does not accept wildcard characters in the output file specification.

16.26.4 CALL Command Defaults

Following are additional defaults associated with using the CALL command:

- You can nest subroutines called with the CALL command and procedures called with the execute procedure (@) command to a maximum of 32 command levels.
- Unless they are masked using the SET SYMBOL command, local symbols defined in an outer level are available to any inner procedure or subroutine levels. Global symbols are available at any command level.
- Labels are valid only for the level in which they are defined.

16.26.5 Beginning and Ending Subroutines

The SUBROUTINE and ENDSUBROUTINE commands define the beginning and the end of a CALL subroutine. The label defining the entry point to the subroutine immediately precedes the SUBROUTINE command. You can place the EXIT command immediately before the ENDSUBROUTINE command but it is not required to terminate the subroutine. The ENDSUBROUTINE command terminates the subroutine and transfers control to the command line immediately following the CALL command.

Command lines in a subroutine execute only when the subroutine is called with the CALL command. During the line-by-line execution of the command procedure, the command language interpreter skips all commands between the SUBROUTINE and the ENDSUBROUTINE commands.

16.26.6 Scope Defining Restrictions

The following restrictions apply to defining the scope of subroutine entry points and the use of label references:

- Subroutine entry points that are defined within another subroutine are local to that subroutine. You cannot call a subroutine if the subroutine entry point is within a separate subroutine block.
- If a subroutine entry point is located within an IF-THEN-ELSE block, you cannot call this subroutine from outside the IF-THEN-ELSE block.
- Every SUBROUTINE command must have a matching ENDSUBROUTINE command to end the subroutine.

16.26.7 Examples: Scope Defining Restrictions

- In the following example, the call is not valid because the CALL BAR command is located outside of the MAIN subroutine:

```
$ CALL BAR
$
$ MAIN: SUBROUTINE
$
$     BAR: SUBROUTINE
$     ENDSUBROUTINE
$
$ ENDSUBROUTINE
```

  For this CALL command to work, it must be placed within the SUBROUTINE and ENDSUBROUTINE points.

- The call shown in this example in not allowed because it is within an IF-THEN-ELSE block:

```
$ IF 1
$ THEN
$    BOB:SUBROUTINE
$    ENDSUBROUTINE
$ ENDIF
$ CALL BOB
```

16.26.8 Example: Calling a Subroutine

The following example includes two subroutines called SUB1 and SUB2. The subroutines do not execute until they are called with the CALL command:

```
$
$! CALL.COM
$
$! Define subroutine SUB1.
$!
$ SUB1: SUBROUTINE
        .
        .
        .
$       CALL SUB2 !Invoke SUB2 from within SUB1.
        .
        .
        .
$       @FILE  !Invoke another command procedure file.
        .
        .
        .
$       EXIT
$ ENDSUBROUTINE !End of SUB1 definition.
$!
$! Define subroutine SUB2.
$!
$ SUB2: SUBROUTINE
$       EXIT
$ ENDSUBROUTINE !End of SUB2 definition.
$!
$! Start of main routine. At this point, both SUB1 and SUB2
$! have been defined but none of the previous commands have
$! been executed.
$!
$ START:
$       CALL/OUTPUT=NAMES.LOG  SUB1 "THIS IS P1"
        .
        .
        .
$       CALL SUB2 "THIS IS P1" "THIS IS P2"
        .
        .
        .
$ EXIT  !Exit this command procedure file.
```

The CALL command invokes the subroutine SUB1 and directs output to the file NAMES.LOG. Subroutine SUB1 calls subroutine SUB2. The procedure executes SUB2, invoking the command procedure FILE.COM with the execute procedure (@) command. When all the commands in SUB1 have executed, the CALL command in the main procedure calls SUB2 a second time. The procedure exits when SUB2 finishes executing.

16.27 Writing Case Statements

16.27.1 Case Statements

A case statement is a special form of conditional code that executes one out of a set of command blocks, depending on the value of a variable or expression. Typically, the valid values for the case statement are labels at the beginning of each command block. The case statement passes control to the appropriate block of code by using the specified value as the target label in a GOTO statement.

To write a case statement, you must:

1. List the labels
2. Write the case statement
3. Write the command blocks

16.27.2 Listing Labels

To list the labels, equate a symbol to a string that contains a list of the labels delimited by slashes (or any character you choose to act as a delimiter). This symbol definition should precede the command blocks.

16.27.3 Example

This is example equates the symbol COMMAND_LIST to the labels PURGE, DELETE and EXIT:

```
$ COMMAND_LIST = "/PURGE/DELETE/EXIT/"
```

16.27.4 Writing the Case Statement

To write the case statement, follow this procedure:

Step	Action
1	Use the INQUIRE command to get the value of the case variable.
2	Use the IF command with F$LOCATE and F$LENGTH to determine whether the value of the case variable is valid.
3	If the case variable is valid, execute the case statement (with a GOTO command) to pass control to the appropriate block of code. Otherwise, display a message and exit or request a different case value.

16.27.5 Example

In the following example, the label is equated to the full command name. Therefore, F$LOCATE includes the delimiters in its search for the command name to ensure that the command is not abbreviated:

```
$GET_COMMAND:
$ INQUIRE COMMAND -
  "Command (EXIT,PURGE,DELETE)"
$ IF F$LOCATE ("/"+COMMAND+"/",COMMAND_LIST) .EQ. -
   F$LENGTH (COMMAND_LIST) THEN GOTO ERROR_1
$ GOTO 'COMMAND'
   .
   .
   .
$ERROR_1:
$ WRITE SYS$OUTPUT "No such command as ''COMMAND'."
$ GOTO GET_COMMAND
```

16.27.6 Writing Command Blocks

Each block of commands may contain one or more commands. Begin each command block with a unique label. End each command block by passing control to a label outside the list of command blocks.

16.27.7 Example

In the following example, each block of commands begins with a unique label (PURGE:, DELETE:) and is ended by passing control to a label (GOTO GET_COMMAND) outside of the current command block:

```
$GET_COMMAND:
   .
   .
   .
$PURGE:
$ INQUIRE FILE
$ PURGE 'FILE'
$ GOTO GET_COMMAND
$ !
$DELETE:
$ INQUIRE FILE
$ DELETE 'FILE'
$ GOTO GET_COMMAND
$ !
$EXIT:
```

16.27.8 Writing Loops

You can write loops that test variables at the beginning of the command block (as described in Chapter 15). However, you can also write loops that test the termination variable at the end of the loop, by following this procedure:

Step	Action
1	Begin the loop.
2	Perform the commands in the body of the loop.
3	Change the termination variable.
4	Test the termination variable. If the condition is not met, go to the beginning of the loop.

Step	Action
5	End the loop.

Note that when you test the termination variable at the end of the loop, the commands in the body of the loop are executed at least once, regardless of the value in the termination variable.

16.27.9 Example

Both of the command blocks shown in this example execute a loop that terminates when COMMAND equals "EX" (EXIT). F$EXTRACT truncates COMMAND to its first two characters. In the first example, COMMAND, the termination variable, is tested at the beginning of the loop; in the second, it is tested at the end of the loop:

```
$ ! EXAMPLE 1
$ !
$GET_COMMAND:
$ INQUIRE COMMAND-
  "Command (EXIT,DIRECTORY,TYPE,PURGE,DELETE,COPY)"
$ COMMAND = F$EXTRACT(0,2,COMMAND)
$ IF COMMAND .EQS. "EX" THEN GOTO END_LOOP
   .
   .
   .
$ GOTO GET_COMMAND
$END_LOOP:
```

```
$ ! EXAMPLE 2
$ !
$GET_COMMAND:
$ INQUIRE COMMAND-
  "Command (EXIT,DIRECTORY,TYPE,PURGE,DELETE,COPY)"
$ COMMAND = F$EXTRACT(0,2,COMMAND)
   .
   .
   .
$ IF COMMAND .NES. "EX" THEN GOTO GET_COMMAND
$ ! End of loop
```

16.27.10 Using Counters

To perform a loop a specific number of times, use a counter as the termination variable. In the following example, 10 file names are input by the user and placed into the local symbols FIL1, FIL2, . . . , FIL10:

```
$ NUM = 1                        ! Set counter
$LOOP:                           ! Begin loop
$ INQUIRE FIL'NUM' "File"        ! Get file name
$ NUM = NUM + 1                  ! Update counter
$ IF NUM .LT. 11 THEN GOTO LOOP ! Test for termination
$END_LOOP:                       ! End loop
   .
   .
   .
```

16.27.11 Example

The following example uses a counter to control the number of times a loop executes. The loop executes 10 times; the termination variable is tested at the end of the loop:

```
$! Obtain 10 file names and store them in the
$! symbols FILE_1 to FILE_10
$!
$ COUNT = 0
$ LOOP:
$    COUNT = COUNT + 1
$    INQUIRE FILE_'COUNT' "File"
$    IF COUNT .LT. 10 THEN GOTO LOOP
$!
$ PROCESS_FILES:
   .
   .
   .
```

The symbol COUNT is used to record the number of times the commands in the loop are executed. COUNT is also used to create the symbol names FILE_1, FILE_2, and so on to FILE_10. Note that the value of COUNT is incremented at the beginning of the loop but is tested at the end of the loop. Therefore, when COUNT is incremented from 9 to 10, the loop executes a last time (obtaining a value for FILE_10) before the IF statement finds a false result.

16.27.12 Using F$ELEMENT

To perform a loop for a known sequence of values, use the F$ELEMENT lexical function. The F$ELEMENT lexical function obtains items from a list of items separated by delimiters. You must supply the item number, the item delimiter, and the list as arguments for F$ELEMENT.

For more information on how to use the F$ELEMENT lexical function, see the *OpenVMS DCL Dictionary*.

16.27.13 Examples

- In the following example, the files CHAP1, CHAP2, CHAP3, CHAPA, CHAPB, and CHAPC are processed in order:

```
$ FILE_LIST = "1,2,3,A,B,C"
$ INDEX = 0
$PROCESS:
$ NUM = F$ELEMENT(INDEX,",",FILE_LIST)
$ IF NUM .EQS. "," THEN GOTO END_LOOP
$ FILE = "CHAP''NUM'"
$ ! process file named by FILE
    .
    .
    .
$ INDEX = INDEX + 1
$ GOTO PROCESS
$END_LOOP:
$ EXIT
```

- In the following example, the command procedure uses a loop to copy the files listed in the symbol FILE_LIST to a directory on another node:

```
$ FILE_LIST = "CHAP1/CHAP2/CHAP3/CHAP4/CHAP5"
$ NUM = 0
$!
$! Process each file listed in FILE_LIST
$ PROCESS_LOOP:
$     FILE = F$ELEMENT(NUM,"/",FILE_LIST)
$     IF FILE .EQS. "/" THEN GOTO DONE
$     COPY 'FILE'.MEM MORRIS::DISK3:[DOCSET]*.*
$     NUM = NUM + 1
$     GOTO PROCESS_LOOP
$!
$ DONE:
$ WRITE SYS$OUTPUT "Finished copying files."
$ EXIT
```

The first file returned by the F$ELEMENT lexical function is CHAP1, the next file is CHAP2, and so on. Each time through the loop, the value of NUM is increased so that the next file name is obtained. When the F$ELEMENT returns a slash, all the items from FILE_LIST have been processed and the loop is terminated.

17

Lexical Functions: Obtaining and Manipulating Information

17.1 Overview

Lexical functions are command language constructs that the DCL interpreter evaluates and substitutes before it interprets a command string. This chapter includes information on:

- How lexical functions work
- Obtaining information about your process
- Obtaining information about the system
- Obtaining information about files and devices
- Translating logical names
- Manipulating strings
- Manipulating data types

17.1.1 References

For additional information on lexical functions, refer to online Help. For additional information on the commands discussed in this chapter, refer to the *OpenVMS DCL Dictionary*.

17.2 About Lexical Functions

17.2.1 Overview

Lexical functions can be used to obtain information about:

- Your process
- The system
- Files and devices
- Logical names
- Strings
- Data types

Many lexical functions return information that you can also get from DCL commands.

17.2.2 Information Manipulation

You can manipulate information in a command procedure more easily if you obtain it from a lexical function rather than from a command. For example, you can use either the F$ENVIRONMENT function or the SHOW DEFAULT command to obtain the name of your current default directory. The differences are as follows:

- If you use the F$ENVIRONMENT function, you can assign the result to a symbol and then use this symbol later in the procedure. For example:

```
$ DIR_NAME = F$ENVIRONMENT("DEFAULT")
$ SET DEFAULT DISK4:[TEST]
   .
   .
   .
$ SET DEFAULT 'DIR_NAME'
```

  The F$ENVIRONMENT function returns the current default disk and directory and stores this value in the symbol DIR_NAME. At the end of the procedure, you use the symbol DIR_NAME to restore the default with the SET DEFAULT command.

- If you obtain the value of the current default directory with the SHOW DEFAULT command, rather than with the F$ENVIRONMENT lexical function, you cannot assign this output to a symbol directly. Instead, the procedure is as follows:

```
$! Redirect the output of the SHOW DEFAULT command to a file.
$ DEFINE/SUPERVISOR_MODE SYS$OUTPUT DISK4:[TEST]TEMPFILE.DAT
$ SHOW DEFAULT
$ DEASSIGN SYS$OUTPUT
$!
$ OPEN/READ DIR_FILE DISK4:[TEST]TEMPFILE.DAT ! Open the file.
$ READ DIR_FILE DIR_NAME,                     ! Read the file.
$ SET DEFAULT 'DIR_NAME'                      ! Reset the directory.
$ CLOSE DIR_FILE                              ! Close the file.
$ DELETE DISK4:[TEST]TEMPFILE.DAT;*           ! Delete the file.
```

17.3 Obtaining Information About Your Process

17.3.1 Process Lexical Functions

You often change process characteristics for the duration of a command procedure and then restore them. You can use the following lexical functions to obtain information about your process:

F$DIRECTORY	Returns the current default directory string.
F$ENVIRONMENT	Returns information about the command environment for your process.

F$GETJPI	Returns accounting, status, and identification information about your process or about other processes on the system.
F$MODE	Shows the mode in which your process is executing.
F$PRIVILEGE	Indicates whether your process has the specified privileges.
F$PROCESS	Returns the name of your process.
F$SETPRV	Sets the specified privileges. This function also indicates whether the specified privileges were previously enabled before you used the F$SETPRV function.
F$USER	Returns your user identification code (UIC).
F$VERIFY	Indicates whether verification is on or off.

17.3.2 Commonly Changed Process Characteristics

The following table shows process characteristics that are commonly changed in command procedures; it also gives the lexical functions that save these characteristics and the DCL commands that restore the original settings.

Characteristic	Operation	Command or Lexical Function
Control characters	Save	F$ENVIRONMENT("CONTROL")
	Restore	SET CONTROL
DCL prompt	Save	F$ENVIRONMENT("PROMPT")
	Restore	SET PROMPT
Default protection	Save	F$ENVIRONMENT("PROTECTION")
	Restore	SET PROTECTION/DEFAULT
Key state	Save	F$ENVIRONMENT("KEY_STATE")
	Restore	SET KEY
Message format	Save	F$ENVIRONMENT("MESSAGE")
	Restore	SET MESSAGE

Characteristic	Operation	Command or Lexical Function
Privileges	Save	F$PRIVILEGE or F$SETPRV
	Restore	F$SETPRV or SET PROCESS /PRIVILEGES
Verification	Save	F$VERIFY or F$ENVIRONMENT
	Restore	F$VERIFY or SET VERIFY

17.3.3 Saving Process Characteristics

If you save process characteristics, you must ensure that an error or Ctrl/Y interruption does not cause the procedure to exit before you restore the original characteristics. (See Chapter 15 for more information on handling errors and Ctrl/Y interruptions.)

17.3.4 Changing Verification Settings

You can use the F$VERIFY lexical function to disable verification for the duration of a command procedure. This prevents users from displaying a procedure's contents while executing the procedure.

There are two types of verification:

- Procedure verification

  Displays each command line as it is being executed

- Image verification

  Displays each data line as it is being processed

By default, the SET [NO]VERIFY command and the F$VERIFY function turn both types of verification on or off. In general, the procedure and image verification settings in a procedure are the same (both on or both off). However, if you decide to change the settings, save each verification setting separately.

17.3.5 Examples

- In the following example, the symbol TEMP is used to enable and disable verification:

```
$ ! Enable verification
$ !
$ TEMP = F$VERIFY(1)
$ LOOP:
$     INQUIRE FILE "File name"
$     IF FILE .EQS."" THEN EXIT
$     PRINT 'FILE'
$     GOTO LOOP
$ ! Disable verification
$ !
$ TEMP = F$VERIFY(0)
$ EXIT
```

- In the following example, the verification settings are saved:

```
$ ! Save each verification state
$ ! Turn both states off
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
    .
    .
    .
$ ! Restore original verification states
$ SAVE_VERIFY_IMAGE = F$VERIFY(SAVE_VERIFY_PROCEDURE,-
      SAVE_VERIFY_IMAGE)
```

  The F$ENVIRONMENT function returns the current image verification setting and assigns this value to the symbol SAVE_VERIFY_IMAGE. Next, the F$VERIFY function returns the current procedure verification setting and assigns this value to the symbol SAVE_VERIFY_PROCEDURE. The F$VERIFY function disables both image and procedure verification. You can use the F$ENVIRONMENT function to obtain the procedure verification setting before you disable verification with F$VERIFY. However, it is shorter to use F$VERIFY to accomplish both tasks in one command line, as shown in the previous example.

  At the end of this procedure, the F$VERIFY function restores the original settings (specified by the symbols SAVE_VERIFY_PROCEDURE and SAVE_VERIFY_IMAGE.)

17.3.6 Time Stamping

If you are using time-stamping, remember that it applies only if verification is enabled. For more information on time-stamping and the SET PREFIX command, see the *OpenVMS DCL Dictionary* or refer to DCL help.

17.3.7 Changing Default File Protection

You may want to change the default file protection within a command procedure. The following command procedure changes the default protection associated with files created while the procedure is executing. The procedure restores the original default file protection before terminating.

```
$ SAVE_PROT = F$ENVIRONMENT("PROTECTION")
$ SET PROTECTION = (SYSTEM:RWED, OWNER:RWED, GROUP, WORLD)/DEFAULT
    .
    .
    .
$ SET PROTECTION=('SAVE_PROT')/DEFAULT
$ EXIT
```

Note that the F$ENVIRONMENT function returns the default protection code using the syntax required by the SET PROTECTION command. This lets you use the symbol SAVE_PROT with the SET PROTECTION command to restore the original default file protection.

17.4 Obtaining Information About the System

17.4.1 System Information

You can use the following lexical functions to obtain information about the system:

F$CONTEXT	Specifies selection criteria to use with the F$PID function. The F$CONTEXT function enables the F$PID function to obtain information about processes from any node in a VMScluster.
F$CSID	Returns a VMScluster identification number and updates the context symbol to point to the current position in the system's VMScluster node list.
F$GETQUI	Returns information about queues, batch and print jobs currently in those queues, form definitions, and characteristic definitions kept in the system job queue file.
F$GETSYI	Returns information about your local system or about a node in your VMScluster (if your system is part of a VMScluster).
F$IDENTIFIER	Converts identifiers from named to numeric format or from numeric to named format.
F$MESSAGE	Returns the message text associated with a status code.
F$PID	Returns the process identification (PID) number for processes that you are allowed to examine.
F$TIME	Returns the current date and time.

17.4.2 Determining Your VMScluster Node Name

If your system is part of a VMScluster environment where you can log in to many different nodes, you can set the DCL prompt to indicate which node you are currently using. To do this, include the F$GETSYI function in your login command procedure to determine the node name. Then use the SET PROMPT command to set a unique prompt for the node.

If you want to use only a portion of the node name in your prompt string, use the F$EXTRACT function to extract the appropriate characters. See Section 17.7.4 for more information on extracting characters.

17.4.3 Example

In the following example, the symbol NODE is defined as FF$GETSYI("NODENAME") and then the node name is used as the prompt:

```
$ NODE = F$GETSYI("NODENAME")
$ SET PROMPT = "''NODE'$ "
    .
    .
    .
```

17.4.4 Obtaining Queue Information

You can use the F$GETQUI function to get many types of information about batch and print queues. You must have read access to the job, SYSPRV privilege, or OPER privilege to obtain information about jobs and files in queues.

17.4.5 Example

The following example shows how to determine if the batch queue VAX1_BATCH is in a stopped state. The value returned is either true or false. If the queue is not stopped, the command procedure submits a job to the queue:

```
$ QSTOPPED = F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", "VAX1_BATCH")
$ IF QSTOPPED THEN GOTO NOBATCH
$ SUBMIT/QUEUE=VAX1_BATCH TEST.COM
$ NOBATCH:
    .
    .
    .
```

17.4.6 Obtaining Process Information

You can use the F$PID function to get the process identification (PID) number for all processes that you are allowed to examine. You can obtain PID numbers:

- For all processes on the system if you have WORLD privilege
- For all processes in your group if you have GROUP privilege
- Only for your process if you have neither GROUP nor WORLD privilege

After you obtain a PID number, you can use the F$GETJPI function to obtain specific information about the process.

17.4.7 Examples

- The following example shows how to obtain and display the PID numbers for the processes you are allowed to examine:

```
$ ! Display the time when this procedure
$ ! begins executing
$ WRITE SYS$OUTPUT F$TIME()
$ !
$ CONTEXT = ""
$ START:
$ ! Obtain and display PID numbers until
$ ! F$PID returns a null string
$ !
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT "Pid --- ''PID'"
$ GOTO START
```

The system uses the symbol CONTEXT to hold a pointer into the system list of PID numbers. Each time through the loop, the system changes the pointer to locate the next PID number in the list. The procedure exits after all PID numbers have been displayed.

- In the following example, the procedure displays the PID number and the UIC for each process:

```
$ CONTEXT = ""
$ START:
$ ! Obtain and display PID numbers and UICs
$ !
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ UIC = F$GETJPI(PID,"UIC")
$ WRITE SYS$OUTPUT "Pid --- ''PID'   Uic--- ''UIC' "
$ GOTO START
```

Note that you can shorten this command procedure by including the F$GETJPI function within the WRITE command, as follows:

```
$ CONTEXT = ""
$ START:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT "Pid --- ''PID'   Uic --- ''F$GETJPI(PID,"UIC")'"
$ GOTO START
```

17.4.8 F$CONTEXT Lexical Function

To obtain information about processes from any node in a VMScluster, use the F$CONTEXT function.

17.4.9 Example

In the following example, F$CONTEXT is called three times to set up selection criteria:

```
$!Establish an error and Ctrl/Y handler
$!
$ ON ERROR THEN GOTO error
$ ON CONTROL_Y THEN GOTO error
$!
$ ctx = ""
❶ $ temp = F$CONTEXT ("PROCESS", ctx, "NODENAME", "*","EQL")
❷ $ temp = F$CONTEXT ("PROCESS", ctx, "USERNAME", "M*,SYSTEM","EQL")
❸ $ temp = F$CONTEXT ("PROCESS", ctx, "CURPRIV", "SYSPRV,OPER", "ALL")
$!
$!Loop over all processes that meet the selection criteria.
$!Print the PID number and the name of the image for each process.
$!
❹ $loop:
$ pid = F$PID(ctx)
$ IF pid .EQS. ""
$ THEN
$     GOTO endloop
$ ELSE
```

```
❺ $     image = F$GETJPI(pid,"IMAGNAME")
  $     SHOW SYMBOL pid
❻ $     WRITE SYS$OUTPUT image
  $     GOTO loop
  $ ENDIF
  $!The loop over the processes has ended.
  $!
  $endloop:
  $!
  $ EXIT
  $!
  $!Error handler. Clean up the context's memory with
  $!the CANCEL selection item keyword.
  $!
  $error:
❼ $ IF F$TYPE(ctx) .eqs. "PROCESS_CONTEXT" THEN -
❽ -$ temp = F$CONTEXT ("PROCESS", ctx, "CANCEL")
  $!
  $ EXIT
```

As you examine the example, note the following:

❶ The first call requests that the search take place on all nodes in the VMScluster.

❷ The second call requests that only the processes whose user name either starts with an M or is SYSTEM be processed.

❸ The third call restricts the selection to those processes whose current privileges include both SYSPRV (system privilege) and OPER (operator) and can have other privileges set.

❹ The command lines between the labels ‘‘loop’’ and ‘‘endloop’’ continually call F$PID to obtain the processes that meet the criteria set up in the F$CONTEXT calls.

❺ After retrieving each PID number, F$GETJPI is called to return the name of the image running in the process.

❻ Finally, the procedure displays the name of the image.

❼ In case of error or a Ctrl/Y operation, control is passed to *error* and the context is closed if necessary.

❽ Note the check for the symbol type PROCESS_CONTEXT. If the symbol has this type, selection criteria must be canceled by a call to F$CONTEXT. If the symbol is not of the type PROCESS_CONTEXT, either selection criteria have not been set up yet in F$CONTEXT or the symbol was used with F$PID until an error occurred or until the end of the process list was reached.

17.5 Obtaining Information About Files and Devices

17.5.1 Overview

You can use the following lexical functions to obtain information about files and devices:

F$DEVICE	Returns the device names of all devices on a system that meet the specified selection criteria
F$FILE_ATTRIBUTES	Returns information about file attributes
F$GETDVI	Returns information about a specified device
F$PARSE	Parses a file specification and returns the requested field or fields
F$SEARCH	Searches a directory for a file

17.5.2 Searching for Devices

To get information on a particular device by using the system service $GETDVI, you must provide the device name to the service. If you do not know the device name, you can find it by using the lexical function F$DEVICE.

The F$DEVICE function allows wildcard searches based on the device name, the device class, or the device type. To do a search based on device type, you must also specify a device class.

You can use the F$DEVICE function in a loop in a command procedure to return device names that match the specified selection criteria. Each time the F$DEVICE function is executed, it returns the next device on the system that matches the selection criteria. Note that devices are returned in no particular order. After the last device name is returned, the next F$DEVICE function call returns a null string.

17.5.3 Example

This command procedure displays the device names of all the RA60s on a unit numbered 0:

```
$ START:
$     DEVICE_NAME = F$DEVICE("*0:","DISK","RA60")
$     IF DEVICE_NAME .EQS. "" THEN EXIT
$     SHOW SYMBOL DEVICE_NAME
$     GOTO START
```

17.5.4 Searching for a File in a Directory

Before processing a file, a command procedure should use the F$SEARCH function to test whether the file exists. For example, the following command procedure uses F$PARSE to apply a device and directory string to the file STATS.DAT. Then the procedure uses the F$SEARCH function to determine whether STATS.DAT is present in DISK3:[JONES.WORK]. If it is, the command procedure processes the file. Otherwise, the command procedure prompts for another input file.

```
$ FILE = F$PARSE("STATS.DAT","DISK3:[JONES.WORK]",,,"SYNTAX_ONLY")
$ IF F$SEARCH(FILE) .EQS. "" THEN GOTO GET_FILE
$ PROCESS_FILE:
   .
   .
   .
$ GET_FILE:
$   INQUIRE FILE "File name"
$   GOTO PROCESS_FILE
```

After determining that a file exists, the procedure can use the F$PARSE or the F$FILE_ATTRIBUTES function to get additional information about the file. For example:

```
$ IF F$SEARCH("STATS.DAT") .EQS. "" THEN GOTO GET_FILE
$ PROCESS_FILE:
$     NAME = F$PARSE("STATS.DAT",, "NAME")
   .
   .
   .
$ GET_FILE:
$   INQUIRE FILE "File name"
$   GOTO PROCESS_FILE
```

17.5.5 Deleting Old Versions of Files

If a command procedure creates files that you do not need after the procedure terminates, delete or purge these files before you exit from the procedure. Use the PURGE command to delete all versions except the most recent one. Use the DELETE command with a version number to delete a specific version of the file or with a wildcard character in the version field to delete all versions of the file.

To avoid error messages when using the DELETE command within a command procedure, use the F$SEARCH function to verify that a file exists before you try to delete it. For example, you can write a command procedure that creates a file named TEMP.DAT only if certain modules are executed. The following line issues the DELETE command only if TEMP.DAT exists:

```
$ IF F$SEARCH("TEMP.DAT") .NES. "" THEN DELETE TEMP.DAT;*
```

17.6 Translating Logical Names

17.6.1 Overview

You can use the following lexical functions to translate logical names:

F$LOGICAL	Returns the equivalence string for a logical name.
F$TRNLNM	Returns either the equivalence string or the requested attributes for a logical name.

17.6.2 Obsolete Function

The F$TRNLNM function supersedes the F$LOGICAL function that was used in earlier versions of the OpenVMS operating system. You should use F$TRNLNM (instead of F$LOGICAL) to ensure that your command procedure processes logical names using the current system techniques.

17.6.3 Variables in Command Procedures

In some situations, you may want to use logical names rather than symbols as variables in command procedures. Programs can access logical names more easily than they can access DCL symbols. Therefore, to pass information to a program that you run from a command procedure, obtain the information using a symbol. Then use the DEFINE command to equate the value of the symbol to a logical name.

You can also use the F$TRNLNM function to assign the value of a logical name to a symbol.

17.6.4 Examples

- The following example tests whether the logical name NAMES has been defined. If it has, the procedure runs PAYROLL.EXE. Otherwise, the procedure obtains a value for the symbol FILE and uses this value to create the logical name NAMES. PAYROLL.EXE uses the logical name NAMES to refer to the file of employee names:

```
$ ! Make sure that NAMES is defined
$ IF F$TRNLNM("NAMES") .NES. "" THEN GOTO ALL_SET
$ INQUIRE FILE "File with employee names"
$ DEFINE NAMES 'FILE'
$ !
$ ! Run PAYROLL, using the file indicated by NAMES
$ ALL_SET:
$ RUN PAYROLL
    .
    .
    .
```

- This command procedure defines a logical name that is used in the program PAYROLL:

```
$ DEFINE NAMES DISK4:[JONES]EMPLOYEE_NAMES.DAT
$ RUN PAYROLL
    .
    .
    .
$ FILE = F$TRNLNM(NAMES)
$ WRITE SYS$OUTPUT "Finished processing ",FILE
```

  At the end of the procedure, the WRITE command displays a message indicating that the file was processed. Because the WRITE command does not translate logical names, you need to equate the logical name (NAMES) to a symbol (FILE). Then you can use the symbol FILE to display the file name.

17.7 Manipulating Strings

17.7.1 Overview

You can use the following lexical functions to manipulate character strings:

F$CVTIME	Returns information about a time string

F$EDIT	Edits a character string
F$ELEMENT	Extracts an element from a string in which the elements are separated by delimiters
F$EXTRACT	Extracts a section of a character string
F$FAO	Formats an output string
F$LENGTH	Determines the length of a string
F$LOCATE	Locates a character or a substring within a string and returns the offset

17.7.2 Determining Presence of Strings or Characters

One common reason for examining strings is to determine whether a character (or substring) is present within a character string. To do this, use the F$LENGTH and the F$LOCATE functions. If the value returned by the F$LOCATE function equals the value returned by the F$LENGTH function, then the character you are looking for is not present.

17.7.3 Example

This procedure requires a file name that includes the version number. To determine whether a version number is present, the procedure tests whether a semicolon (;), which precedes a version number in a file name, is included in the file name that the user enters:

```
$ INQUIRE FILE "Enter file (include version number)"
$ IF F$LOCATE(";", FILE) .EQ. F$LENGTH(FILE) THEN -
     GOTO NO_VERSION
   .
   .
   .
```

The F$LOCATE function returns the offset for the semicolon. Offsets start with 0; thus, if the semicolon were the first character in the string, the F$LOCATE function would return the integer 0. If the semicolon is not present within the string, the F$LOCATE function returns an offset that is one more than the offset of the last character in the string. This value is the same as the length returned by F$LENGTH, which measures the length of the string starting with the number 1.

17.7.4 Extracting Parts of Strings

To extract a portion of a string, use either the F$EXTRACT function or the F$ELEMENT function. Use the F$EXTRACT function to extract a substring that starts at a defined offset. Use the F$ELEMENT function to extract part of a string between two delimiters. In order to use either of these functions, you must know the general format of the string you are parsing. Note that you do not need to use F$EXTRACT or F$ELEMENT to parse file specifications or time strings. Instead, use F$PARSE or F$CVTIME to extract the desired portions of file specifications or time strings.

You can also determine the length of the group name at the same time you extract it.

If a string contains a delimiter that separates different parts of the string, use the F$ELEMENT function to extract the part that you want. You can use F$ELEMENT to obtain different types of access by extracting the portions of the string between the commas. To determine system access, obtain the first element; to determine owner access, obtain the second element; and so on. Note that when you use the F$ELEMENT function, element numbers start with zero. For this reason, use the integer 3 to specify the fourth element.

17.7.5 Example

- This command procedure uses the F$EXTRACT function to extract the group portion of the UIC. This allows the procedure to execute a different set of commands depending on the user's UIC group:

```
$ UIC = F$USER()
$ GROUP_LEN = F$LOCATE(",",UIC) - 1
$ GROUP = F$EXTRACT(1,GROUP_LEN, UIC)
$ GOTO 'GROUP'_SECTION
   .
   .
   .
$ WRITERS_SECTION:
   .
   .
   .
$ MANAGERS_SECTION:
   .
   .
   .
```

  First, the procedure determines the UIC with the F$USER function. Next, the procedure determines the length of the group name by using F$LOCATE to locate the offset of the comma. The comma separates the group from the user portion of a UIC. Everything between the left bracket and the comma is part of the group name. For example, the group name from the UIC [WRITERS,SMITH] is WRITERS.

  After determining the length, the procedure extracts the name of the group with the F$EXTRACT function. The name starts with offset 1 and ends with the character before the comma. Finally, the procedure directs execution to the appropriate label.

- In the following example shows how to determine the length of a group name at the same time it is being extracted:

```
$ UIC = F$USER()
$ GROUP = F$EXTRACT(1, F$LOCATE(",",UIC) - 1, UIC)
$ GOTO 'GROUP'_SECTION
```

- The following example shows how each type of access in a protection codes is separated by a comma:

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ SHOW SYMBOL PROT
PROT = "SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD"
```

- The commands in this example extract the world access portion (the fourth element) from the default protection code:

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ WORLD_PROT = F$ELEMENT(3,",",PROT)
   .
   .
   .
```

  The F$ELEMENT function returns everything between the third comma and the end of the string. Thus, if your default protection allowed read access for world users, the string "WORLD=R" would be returned.

  After you obtain the world access string, you may need to examine it further. For example:

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ WORLD_PROT = F$ELEMENT(3,",",PROT)
$ IF F$LOCATE("=", WORLD_PROT) .EQ. F$LENGTH(WORLD_PROT) -
  THEN GOTO NO_WORLD_ACCESS
    .
    .
    .
```

17.7.6 Formatting Output Strings

You can use the WRITE command to write a string to a record. To line up columns in a record, you can use the F$FAO function to define record fields and place the process name and user name in these fields. When you use the F$FAO function, use a control string to define the fields in the record; then specify the values to be placed in these fields.

Another way to format fields in a record is to use a character string overlay. Note, however, that the F$FAO function is more powerful than a character string overlay. You can perform a wider range of output operations with the F$FAO function.

17.7.7 Examples

- The command procedure shown in this example uses the WRITE command to display the process name and PID number for processes on the system:

```
$ ! Initialize context symbol to get PID numbers
$ CONTEXT = ""
$ ! Write headings
$ WRITE SYS$OUTPUT "Process Name     PID"
$ !
$ GET_PID:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT F$GETJPI(PID,"PRCNAM"),"     ", F$GETJPI(PID,"PID")
$ GOTO GET_PID
```

Note that the output from the WRITE command inserts five spaces between the process name and the user name but the columns do not line up:

```
Process Name     PID
MARCHESAND     2CA0049C
TRACTMEN     2CA0043A
FALLON     2CA0043C
ODONNELL     2CA00453
PERRIN     2CA004DE
CHAMPIONS     2CA004E3
```

- The command procedure in this example uses the F$FAO function to define a 16-character field and a 12-character field. The F$FAO function places the process name in the first field, skips a space, and then places the PID number in the second field:

```
$ ! Initialize context symbol to get PID numbers
$ CONTEXT = ""
$ ! Write headings
$ WRITE SYS$OUTPUT "Process Name     PID"
$ !
$ GET_PID:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ LINE = F$FAO("!16AS !12AS", F$GETJPI(PID,"PRCNAM"), F$GETJPI(PID,"PID"))
$ WRITE SYS$OUTPUT LINE
$ GOTO GET_PID
```

Now when you execute the procedure, the columns will align:

```
Process Name     PID
MARCHESAND       2CA0049C
TRACTMEN         2CA0043A
FALLON           2CA0043C
ODONNELL         2CA00453
PERRIN           2CA004DE
CHAMPIONS        2CA004E3
```

- The following example uses an overlay to place the process name in the first 16 characters (starting at offset 0) of the symbol RECORD. Then the PID number is placed in the next 12 characters (starting at offset 17):

```
$ ! Initialize context symbol to get PID numbers
$ CONTEXT = ""
$ ! Write headings
$ WRITE SYS$OUTPUT "Process Name     PID"
$ !
$ GET_PID:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ RECORD[0,16]:= 'F$GETJPI(PID,"PRCNAM")'
$ RECORD[17,12]:= 'F$GETJPI(PID,"PID")'
$ WRITE SYS$OUTPUT RECORD
$ GOTO GET_PID
```

This procedure produces the same type of formatted columns you created with the F$FAO function:

```
Process Name     PID
MARCHESAND       2CA0049C
TRACTMEN         2CA0043A
FALLON           2CA0043C
ODONNELL         2CA00453
PERRIN           2CA004DE
CHAMPIONS        2CA004E3
```

17.8 Manipulating Data Types

17.8.1 Overview

You can use the following lexical functions to convert data from strings to integers and from integers to strings:

F$CVSI	Extracts bit fields from a character string and converts the result, as a signed value, to an integer
F$CVUI	Extracts bit fields from a character string and converts the result, as an unsigned value, to an integer
F$INTEGER	Converts a string expression to an integer
F$STRING	Converts an integer expression to a string
F$TYPE	Determines the data type of a symbol

17.8.2 Converting Data Types

Use the F$INTEGER and F$STRING functions to convert between integers and strings. For example, the following command procedure converts data types. If you enter a string, the command procedure shows the integer equivalent. If you enter an integer, the command procedure shows the string equivalent. Note how the F$TYPE function is used to form a label name in the GOTO statement; F$TYPE returns ‘‘STRING’’ or ‘‘INTEGER’’ depending on the data type of the symbol.

```
$ IF P1 .EQS. "" THEN INQUIRE P1 "Value to be converted"
$ GOTO CONVERT_'F$TYPE(P1)'
$
$ CONVERT_STRING:
$ WRITE SYS$OUTPUT "The string ''P1' is converted to ''F$INTEGER(P1)'"
$ EXIT
$
$ CONVERT_INTEGER:
$ WRITE SYS$OUTPUT "The integer ''P1' is converted to ''F$STRING(P1)'"
$ EXIT
```

17.8.3 Evaluating Expressions

Some commands, such as INQUIRE and READ, accept only string data. If you use these commands to obtain data that you want to evaluate as an integer expression, use the F$INTEGER function to convert and evaluate this data.

Note that you must place apostrophes (`' '`) around the symbol EXP when you use it as an argument for the F$INTEGER function. This causes DCL to substitute the value for EXP during the first phase of symbol substitution.

17.8.4 Example

In the following example, the F$INTEGER function is used to evaluate an integer expression:

```
$ INQUIRE EXP "Enter integer expression"
$ RES = F$INTEGER('EXP')
$ WRITE SYS$OUTPUT "Result is",RES
```

The output from this command procedure would be as follows:

```
Enter integer expression: 9 + 7
Result is 16
```

The value ''9 + 7'' is substituted. When the F$INTEGER function processes the argument ''9 + 7,'' it evaluates the expression and returns the correct result.

17.8.5 Determining Whether a Symbol Exists

Use the F$TYPE function to determine whether a symbol exists. The F$TYPE function returns a null string if a symbol is undefined. For example:

```
   .
   .
   .
$ IF F$TYPE(TEMP) .EQS. "" THEN TEMP = "YES"
$ IF TEMP .EQS. "YES" THEN GOTO TEMP_SEC
   .
   .
   .
```

This procedure tests whether the symbol TEMP has been previously defined. If it has, then the current value of TEMP is retained. If TEMP is not defined, then the IF statement assigns the value "YES" to TEMP.

18

Processes and Batch Jobs: Using the OpenVMS Environment

18.1 Overview

A process is an environment created by the OpenVMS operating system that lets you interact with the system. This chapter describes:

- Interpreting your process context
- Using subprocesses
- Connecting to disconnected processes on virtual terminals
- Working with batch jobs

18.1.1 Resources

For additional information on the commands described in this chapter, refer to online Help or the *OpenVMS DCL Dictionary*.

18.1.2 How Processes Are Created

The system creates a process for you when you perform one of the following tasks:

- Logging in

  The system creates a process for each interactive user.
- Submitting a batch job

  The system creates a process for each batch job. When the batch job is completed, the system deletes the process.
- Spawning a subprocess

  The system creates a process when you use the SPAWN command.
- Running a program

  The system creates a process when you run a program using either the /DETACHED qualifier or the /UIC=uic qualifier.

18.2 Interpreting Your Process Context

18.2.1 Displaying Process Context

Characteristics that a process uses, such as privileges, symbols, and logical names, form a process context. To display the process context for your current process, enter the SHOW PROCESS/ALL command.

18.2.2 Example

The following example shows a process context:

```
11-DEC-1995 13:30:37.12 ❶  User: CLEAVER ❷  Process ID: 24E003DC    ❸
                           Node: ZEUS        Process name: CLEAVER_1
Terminal:                                                          ❹
User Identifier:    [DOC,CLEAVER]   ❺
Base priority:      4               ❻
Default file spec:  DISK1:[CLEAVER] ❼

Process Quotas:                     ❽
 Account name: DOC
 CPU limit:                       Infinite  Direct I/O limit:        18
 Buffered I/O byte count quota:      31808  Buffered I/O limit:      25
 Timer queue entry quota:               10  Open file quota:         57
 Paging file quota:                  22276  Subprocess quota:         4
 Default page fault cluster:            64  AST quota:               38
 Enqueue quota:                        600  Shared file limit:        0
 Max detached processes:                 0  Max active jobs:          0

Accounting information:             ❾
 Buffered I/O count:         140  Peak working set size:        383
 Direct I/O count:             7  Peak virtual size:           2336
 Page faults:                304  Mounted volumes:                0
 Images activated:             1
 Elapsed CPU time:       0 00:00:00.55
 Connect time:           0 00:00:22.76

Process privileges:                 ❿
 GROUP                 may affect other processes in same group
 TMPMBX                may create temporary mailbox
 OPER                  operator privilege
 NETMBX                may create network device

Process rights identifiers:         ⓫
 INTERACTIVE
 LOCAL
 SYS$NODE_ZEUS

Process Dynamic Memory Area         ⓬
  Current Size (bytes)        25600   Current Total Size (pages)    50
  Free Space (bytes)          19592   Space in Use (bytes)        6008
  Size of Largest Block       19520   Size of Smallest Block        24
  Number of Free Blocks           3   Free Blocks LEQU 32 Bytes      1

Processes in this tree:             ⓭
CLEAVER
  CLEAVER_1 (*)
```

As you examine the example, note the following:

❶ Current date and time

The date and time when the SHOW PROCESS/ALL command is executed.

❷ User name

The user name assigned to the account that is associated with the process.

❸ Process identification (PID) number

A unique number assigned to the process by the system. The SHOW PROCESS command displays the PID number as a hexadecimal number.

❹ Process name

The name assigned to the process. Because process names are unique (within a specific UIC group), the first process logged in under an account is assigned the user name. Subsequent processes logged in under the same account are assigned the terminal name. You can change your process name with the DCL command SET PROCESS/NAME.

❺ User identification code (UIC)

The group and member numbers (or letters) assigned to the account that is associated with the process (for example, [DOC,CLEAVER]). Part of your UIC identifies the group to which you belong. Within a group, users are allowed to share files or system resources more freely than between groups.

❻ Priority

The current priority of the process.

❼ Default file specification

The current device and directory. Change your current defaults with the DCL command SET DEFAULT.

❽ Process quotas

The quotas (limits) associated with the process. Examine these quotas with the /QUOTAS or /ALL qualifiers of the SHOW PROCESS command.

❾ Accounting information

The continuously updated account of the process' use of memory and CPU time. Examine this information with the /ACCOUNTING or /ALL qualifiers of the SHOW PROCESS command.

❿ Process privileges

The privileges granted to your processes. Privileges restrict the performance of certain system activities to certain users. Examine your privileges with the /PRIVILEGES or /ALL qualifiers of the SHOW PROCESS command.

⓫ Process rights identifiers

System-defined identifiers that are used in conjunction with access control list (ACL) protection. Identifiers provide the means of specifying the users in an ACL. An ACL is a security tool that defines the kinds of access to be granted or denied to users of an object, such as a file, device, or mailbox.

⓬ Process dynamic memory area

The process' current use of dynamic memory. Dynamic memory is allocated by the system to an image when that image is executing. When that memory is no longer needed by one process, the system allocates it to another process.

Examine this information with the /MEMORY or /ALL qualifiers of the SHOW PROCESS command.

⓭ Processes in this tree

A list of subprocesses belonging to the parent process. An asterisk (*) appears after the current process. Examine this list with the SHOW PROCESS/SUBPROCESSES or /ALL command.

18.2.3 Using Detached Processes

A detached process is either interactive or noninteractive, depending on the parent process. Either you or the operating system perform the login, depending on the argument you provided to the DCL command RUN or the Create Process system service ($CREPRC). (Both RUN and $CREPRC execute the LOGINOUT.EXE image in SYS$SYSTEM.)

18.3 Using Subprocesses

18.3.1 Overview

The SPAWN command enables you to create a subprocess of your current process. Within this subprocess, you can interact with the system and log out of the subprocess to return to your parent process or switch between your parent process and subprocesses. Only one of your processes is executing at any time.

18.3.2 Job Trees

Each user on the system is represented by a **job tree.** A job tree is a hierarchy of all your processes and subprocesses with your main process at the top. A subprocess is dependent on the parent process and is deleted when the parent process exits. By default, the subprocess assumes the name of the parent process followed by an underscore and a unique number. For example, if the parent process name is DOUGLASS, the subprocesses are named DOUGLASS_1, DOUGLASS_2, and so on.

18.3.3 Using Subprocesses to Spawn Tasks

To interrupt a task, perform a second task, then return to the original task, you can use Ctrl/Y to interrupt the first task, spawn a subprocess to perform the second task, exit from the subprocess, and then enter the CONTINUE command to return to the first task. By default, when you create a subprocess, the parent process hibernates and you are given control at DCL level within the subprocess. Your default directory is the current directory of the parent process. For example, if you press Ctrl/Y to interrupt an EVE editing session, enter the CONTINUE command and press Ctrl/W to refresh the screen.

18.3.4 Using Subprocesses to Perform Multiple Tasks

To perform a second task while continuing to work on your original task, you can create the subprocess with the SPAWN/NOWAIT command. SPAWN/NOWAIT generates a noninteractive, batch-like subprocess and is used to execute only commands that do not require input.

Because both the parent and the subprocess are executing concurrently, both attempt to control the terminal. To prevent conflicts, also specify the following:

- /OUTPUT qualifier

  Indicates that the subprocess should write output to a specified file rather than to the terminal

- SPAWN command parameter or /INPUT qualifier

  Indicates that the subprocess should execute the specified commands rather than reading input from the terminal

When you specify the /INPUT qualifier of the SPAWN command, the subprocess is created as a noninteractive process that exits upon encountering a severe error or an end-of-file indicator. At DCL level, Ctrl/Z is treated as an end-of-file indicator.

18.3.5 Creating a Subprocess

Because each process you create is unique, commands executed in one process do not usually affect any other process. However, because control of the terminal passes between processes, commands that affect the terminal characteristics (for example, SET TERMINAL) affect any process controlling that terminal. For example, if one process inhibits echoing and exits without restoring it, echoing remains inhibited for the next process that gains control of the terminal. Reset any altered terminal characteristics with the SET TERMINAL command.

18.3.6 Example

In the following example, a user interrupts a command image (the TYPE command) by pressing Ctrl/Y, spawns a subprocess, exits from the subprocess, and returns to the original process:

```
$ TYPE MICE.TXT
Once the weather turns cold, mice may find a crack in the
foundation and enter your house.  They are looking for food and
shelter from the harsh weather ahead.
   .
   .
   .
Ctrl/Y
$ SPAWN
%DCL-S-SPAWNED, process DOUGLASS_1 spawned
%DCL-S-ATTACHED, terminal now attached to process DOUGLASS_1
$ MAIL
MAIL>
   .
   .
   .
MAIL> EXIT
$ LOGOUT
  Process DOUGLASS_1 logged out at 31-DEC-1995 12:42:12.46
%DCL-S-RETURNED, control returned to process DOUGLASS
$ CONTINUE
Once inside, they may gnaw through electrical wires and raid
your food. Because mice reproduce so quickly, what started
as one or two mice can quickly become an invasion.  If you seal
the cracks and holes on the exterior of your foundation, you can
prevent these rodents from ever getting in.
```

18.3.7 Exiting from a Subprocess

To exit from a subprocess created by the SPAWN command, use one of the following commands:

- LOGOUT

  When you exit from a subprocess with the LOGOUT command, the subprocess is deleted (along with any subprocesses that it created) and you are returned to the parent process.

- ATTACH

  When you exit from a subprocess with the ATTACH command, the subprocess hibernates and control of your terminal is transferred to the specified process. You must specify either a process name as a parameter to the ATTACH command or a process identification (PID) number as a value of the /IDENTIFIER qualifier of the ATTACH command.

18.3.8 Example

The following example shows how to exit from the subprocess DOUGLASS_1 and attach to the process DOUGLASS:

```
$ ATTACH DOUGLASS

%DCL-S-RETURNED, control returned to process DOUGLASS

$ SHOW PROCESS

11-DEC-1995 10:34:58.50   VTA303             User: DOUGLASS
Pid: 25C002B4   PROG. name: DOUGLASS         UIC: [200,200]
Priority:   4   Default file spec: SYS$SYSDEVICE:[DOUGLASS]

Devices allocated: $11$VTA303:
```

18.3.9 Subprocess Context

The subprocess context is the environment that the subprocess inherits from the parent process. By default, a subprocess inherits the following items: defaults, privileges, symbols, logical names, control characters, message format, verification state, and key definitions. Collectively, these items create an environment for the subprocess.

18.3.10 Parent Process Characteristics Not Inherited by the Subprocess

The following items are not inherited from parent processes:

- Process identification (PID) number

  The system assigns each created subprocess a unique PID number.

- Process name

  By default, the subprocess name consists of the name of the parent process followed by an underscore and an integer. Use the /PROCESS qualifier of the SPAWN command to specify a process name other than the default. A process name must be unique.

- Created commands

  Commands that are defined by a parent process using the SET COMMAND command are not copied to a subprocess. To use a created command in a subprocess, you must use SET COMMAND to create that command for the subprocess.

- Authorize privileges

  When you spawn to a subprocess, the process context contains the privileges of the parent process, not the privileges that you are authorized to enable. For example, if you plan to spawn to a subprocess while in Mail and to perform a privileged operation, you must first set the proper privilege in the parent process before you invoke Mail.

18.3.11 Preventing Inheritance

You can use the following SPAWN command qualifiers to prevent the subprocess from inheriting a number of these items:

SPAWN Command Qualifier	Items Inhibited or Changed
/CARRIAGE_CONTROL, /PROMPT	DCL prompt
/NOCLI	CLI (command language interpreter; DCL by default)
/NOKEYPAD	Keypad definitions
/NOLOGICAL_NAMES	Logical names
/NOSYMBOL	Symbols

The /SYMBOL and /LOGICAL_NAMES qualifiers do not affect system-defined symbols (such as $SEVERITY and $STATUS)

or system-defined logical names (such as SYS$COMMAND and SYS$OUTPUT).

18.3.12 Transfer of Control

Because copying logical names and symbols to a subprocess can be time-consuming (a few seconds), you may want to use the /NOLOGICAL_NAMES and /NOSYMBOL qualifiers to the SPAWN command unless you plan to use the logical names or symbols in the subprocess. If you use subprocesses frequently, the ATTACH command provides the most efficient way to enter and exit a subprocess. This method allows you to transfer control quickly between the parent process and subprocess rather than repeatedly waiting for the system to create a new subprocess for you.

18.4 Connecting to Disconnected Processes on Virtual Terminals

18.4.1 Overview

If virtual terminals are enabled and a modem line connection is lost, a process remains active on the system as a disconnected virtual terminal process. You must reconnect to the process within the time period specified by the system manager (the default value is 900 seconds or 15 minutes). If you fail to reconnect to the process before this time expires, the system deletes the process.

Note

You can connect only to a virtual terminal process associated with your user identification code (UIC).

18.4.2 Terminal Disconnections

A terminal can be disconnected in the following circumstances:

- You lose the modem signal between the host and the terminal.
- You press the BREAK key on a terminal with the TT2$M_SECURE characteristic set.
- You enter the DCL command DISCONNECT.
- You enter the DCL command CONNECT/CONTINUE.

18.4.3 Process Reconnections

If your process is disconnected, you have the option of reconnecting to the old process and returning to the state it was in before you were disconnected. When you log in, the system prompts you as follows:

```
    You have the following disconnected process:
Terminal   Process name    Image name
VTA52:     RWOODS          (none)
Connect to above listed process [YES]:
```

If you press the Return key or enter Yes, you are logged out of your current process as if automatic execution of the DCL command CONNECT/CONTINUE had been performed for you. If you enter No or if you delay too long in responding (so that a response period timeout occurs), you remain logged in to your new process. You lose the ability to connect to the old process.

When you have multiple disconnected sessions, you are prompted for the name of the virtual terminal to which you want to reconnect. If you do not want to connect to any of the displayed sessions, enter No.

18.4.4 Removing Disconnected Processes

The system automatically removes your disconnected processes after a certain interval. You can conserve system resources, however, if you directly log out of any disconnected processes, as follows:

Step	Task
1	Enter the DCL command SHOW USERS to determine if you have other disconnected jobs.
2	Enter the DCL command CONNECT/LOGOUT to log out of the current process. Connect back through each of the associated virtual terminals (as noted by the terminal prefix VTA) until you reach the last existing process.
3	Enter the DCL command LOGOUT.

18.4.5 Managing Disconnected Processes

Virtual terminals allow you to maintain more than one disconnected process at a time. You must keep in mind, however, that while you are logged in to a virtual terminal, the physical terminal is disconnected. Any I/O requests directed to a device other than the physical terminal associated with your current virtual terminal process will enter a waiting state. The pending process will terminate when the timeout period expires. If, however, you reconnect to the physical terminal that is to receive the I/O request, the process continues from the point at which it entered the waiting state. Naming each process with a name that relates to its context makes it easier to reconnect to the desired process.

For example, a user named SMITH running a process to edit a file might use the SET PROCESS/NAME command to name the process SMITH_EDIT. Later, to continue editing, SMITH can easily determine which process is appropriate.

A system manager can restrict the use of virtual terminals systemwide or on a per terminal basis.

18.5 Working with Batch Jobs

18.5.1 Overview

A batch job is a noninteractive process. Because a batch job executes in a process of its own, you can have two or more processes doing different things at the same time. For example, you can use batch jobs to:

- Perform a task interactively while the system executes a program or command procedure in batch mode
- Run command procedures that take a long time to execute
- Execute command procedures or programs after hours
- Run certain programs at a reduced priority (for example, if the program uses a disproportionate amount of system resources)

18.5.2 Submitting Batch Jobs

When you submit a batch job, the system creates a detached process with your account and process characteristics. The system runs the job from that process and deletes the process when the job completes. The system also executes the system login command procedure (SYLOGIN.COM) and your login command procedure (LOGIN.COM) and then executes the command procedures in the batch job. As these procedures execute, output is written to a log file. When the batch job completes, you can print the log file or save it in one of your directories.

18.5.3 Using the SUBMIT Command

To run a job in batch mode, submit your job to a batch queue (a list of batch jobs waiting to execute) by entering the DCL command SUBMIT. When you submit a job, it is directed to the default batch queue SYS$BATCH where it is added to the end of the queue of jobs waiting to be executed. When the jobs preceding yours are completed, your job is executed. On an OpenVMS system, the number of batch jobs that can execute simultaneously is specified when the batch queue is created by the system manager. By default, the SUBMIT command uses a file type .COM.

18.5.4 Example

In the following example, the command enters JOB1.COM into SYS$BATCH:

```
$ SUBMIT JOB1
Job JOB1 (queue SYS$BATCH, entry 651, started on SYS$BATCH)
```

18.5.5 Job Entries

The system displays the name of the job, the queue containing the job, and the entry number assigned to the job. You receive the DCL prompt after your job is submitted to the batch queue. If you need to reference your batch job in any DCL commands (DELETE /ENTRY, for example), do so by using the job entry number. (You can obtain the job entry number by using the SHOW ENTRY command.) Note that if multiple procedures are submitted in a

batch job, the batch job terminates when any procedure exits with an error or fatal (severe) system message.

18.5.6 Specifying Start Times

Your batch job does not necessarily have to start running at the time you submit it to the batch queue. To specify a different time, enter the SUBMIT/AFTER command. In the following example, the job is submitted after 11:30 P.M.:

```
$ SUBMIT/AFTER=23:30 JOB1.COM
```

18.5.7 Ensuring Correct Access of Files

When you submit a command procedure for batch execution, the system saves the complete file specification for the command procedure, including the version number. If you update a command procedure after you submit it, the batch job executes the version of the command procedure that you submitted rather than the new version.

Because your login defaults are not usually the defaults needed to access the files mentioned in your command procedures, use one of the following methods to ensure that the correct files are accessed:

- Use complete file specifications—When referring to a file in a command procedure or when passing a file to a command procedure, include the device and directory names as part of the file specification.
- Use the SET DEFAULT command—Before referring to a file in a command procedure, use the SET DEFAULT command at the beginning of the command procedure to specify the proper device and directory.

As a batch job executes, it writes output to a log file. By default, the log file has the same name as the command procedure you submit with the file type .LOG. When the job is finished, the system prints the log file and deletes it from your directory. See Section 18.5.15 for information on saving log files.

18.5.8 Checking for Batch Jobs in Your Login Command Procedure

Each time you submit a batch job, the system executes your login command procedure. You can cause sections of your login command procedure to be included or omitted when you execute batch jobs by using the F$MODE lexical function to test for batch jobs.

18.5.9 Example

In the following example, the login command procedure includes commands, logical names, and symbols that are used exclusively for batch jobs. The section is labeled BATCH_COMMANDS, and the following command is included at the beginning of the login command procedure:

```
IF F$MODE() .EQS. "BATCH" THEN GOTO BATCH_COMMANDS
    .
    .
    .
```

To prevent the system from executing any commands in your login command procedure when you submit a batch job, place the following command at the beginning of the procedure:

```
IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
```

You can place this command anywhere in your login command procedure. When you submit a batch job, the system executes your login command procedure only to the point at which the preceding command is placed.

18.5.10 Submitting Multiple Command Procedures

When you enter the SUBMIT command, you can specify several command procedures to be executed in one job. Unless you specify a name with the /NAME qualifier, the SUBMIT command uses the name of the first command procedure as the job name. Note that if an error causes any command procedure in a job to exit, the entire job terminates.

When a batch job executes, the operating context of the first procedure (UPDATE.COM) is not preserved for the second procedure (SORT.COM). The system deletes local symbols created by UPDATE.COM before SORT.COM executes. Global symbols, however, are preserved.

You cannot specify different parameters for individual command procedures within a single job.

18.5.11 Examples

- In the following example, the SUBMIT command creates a batch job that executes UPDATE.COM then SORT.COM:

```
$ SUBMIT UPDATE,SORT
Job UPDATE (queue SYS$BATCH, entry 207) started on SYS$BATCH
```

- The following example passes the same two parameters to UPDATE.COM and to SORT.COM:

```
$ SUBMIT UPDATE, SORT/PARAMETERS = -
_$ (DISK1:[ACCOUNT.BILLS]DATA.DAT, DISK2:[ACCOUNT]NAME.DAT)
$ Job UPDATE (queue SYS$BATCH, ENTRY 208) started on SYS$BATCH
```

18.5.12 Passing Data to Batch Jobs

The default input stream (SYS$INPUT) for a batch job is the command procedure that is being executed. Because a detached process is executing the batch job, you cannot redefine SYS$INPUT to the terminal (as you can with command procedures that you execute interactively.) To pass input to a batch job, use one of the following techniques:

- Include the data in the command procedure itself.

  To include data in a command procedure, place the data on the lines after the command or image.

- Temporarily define SYS$INPUT as a file.

  To define SYS$INPUT temporarily as a file, use the DEFINE /USER_MODE command.

- Pass parameters to the command procedure when you submit it for execution.

  To pass parameters to a command procedure, use the /PARAMETERS qualifier when you submit the batch job.

Note that you cannot specify different parameters for individual command procedures within a single job. Use separate SUBMIT commands if you need to pass different groups of parameters.

18.5.13 Examples

- In the following example, data lines are passed to the image AVERAGE.EXE:

```
$! Execute AVERAGE.EXE
$ RUN AVERAGE
647
899
532
401
$ EXIT
```

- In the following example, SYS$INPUT is temporarily defined as a file:

```
$ DEFINE/USER_MODE SYS$INPUT STATS.DAT
$ RUN AVERAGE
$ EXIT
```

- In the following example, the parameters in the file EMPLOYEES.DAT are passed to the command procedure CHECKS.DAT:

```
$ SUBMIT/PARAMETERS=(DISK1:[PAYROLL]EMPLOYEES.DAT) CHECKS
Job CHECKS (queue SYS$BATCH, entry 209) started on SYS$BATCH
```

18.5.14 Passing Confidential Information

The SHOW QUEUE/FULL command displays full information about jobs in a batch queue. This display includes any parameters that you pass to the procedure. Therefore, do not pass confidential information (such as a password) to a batch job.

18.5.15 Control of Batch Job Output

By default, the log file has the same name as the first command procedure in the batch job and has the file type .LOG. The system writes output from a batch job to a log file once each minute. To specify a different time interval, include the SET OUTPUT_RATE command in your command procedure.

If you attempt to use the EDT editor to read the log file while the system is writing to it, you receive a message indicating that the file is locked by another user. Wait a few seconds and try again. The EVE editor, however, allows you to read the batch job's log file. By specifying EDIT/TPU/READ_ONLY and the name of the log file, you can use EVE commands to move around the log file and to ensure that any changes you make to the file are not saved. If you omit the /READ_ONLY qualifier and modify the log file in any way, the batch job terminates.

18.5.16 Log File Output

Because your batch job is a process that logs in under your user name and executes your login command procedure, the output from a batch job includes the contents of your login command procedure. The output also includes everything written to the batch job log file (command procedure output, error messages, and so on) and the full logout message. To prevent your login command procedure from being written to the batch log file, add the following command to the beginning of your login command procedure:

```
$ IF F$MODE() .EQS. "BATCH" THEN SET NOVERIFY
```

18.5.17 Modifying Log File Names

By default, the log file name is the name under which you submitted the job. Also by default, the log file has the file type .LOG and assumes the device and directory specified by your login defaults. To specify a different log file name when you submit the job, use the /LOG_NAME qualifier to the SUBMIT command.

18.5.18 Preventing Printing

The batch job log file includes all output to SYS$OUTPUT and SYS$ERROR. It also includes, by default, all command lines executed in the command procedure. To prevent the command lines from being printed, use either the SET NOVERIFY command or the F$VERIFY lexical function in your command procedure. When the job completes, the system writes job termination information (using the long form of the system logout message) to the log file.

18.5.19 Time Stamping

If the SET VERIFY command is in effect, you can also learn the exact time when each command is executed by using the SET PREFIX command to **time-stamp** each command line.

When a batch job fails to complete successfully, you can examine the log file to determine the point at which the command procedure failed and the error status that caused the failure.

18.5.20 Saving Log Files

To save log files, use either the /KEEP or the /NOPRINTER qualifier. The /KEEP qualifier saves the log file after it is printed. The /NOPRINTER qualifier saves the log without printing it. If you specify neither of these qualifiers, the default action occurs; the log file is queued to the default print queue SYS$PRINT and is deleted after it prints. The /KEEP and /NOPRINTER qualifiers save the log file in your default login directory. The log file has the same name as the first command procedure in the batch job and the file type .LOG. To specify an alternate file name or directory name, or both, use the /LOG_FILE qualifier. To rename and save the log file, you must use /LOG_FILE and either /KEEP or /NOPRINTER.

18.5.21 Example

In the following example, the log file is saved to a file named DISK2:[JONES.RESULTS]UPDATE.LOG:

```
$ SUBMIT/LOG_FILE=DISK2:[JONES.RESULTS]/NOPRINTER -
_$ DISK2:[JONES.RESULTS]UPDATE
```

18.5.22 Reading the Log File

You can use the TYPE command to read the log file to determine how much of the batch job has completed. However, if you attempt to display the log file while the system is writing to it, you receive a message indicating that the file is locked by another user. If this occurs, wait a few seconds and try again.

18.5.23 Including Command Output in the Batch Job Log

Typically, a batch job command procedure that compiles, links, and executes a program creates additional printed output such as a compiler listing or a linker map. To produce printed copies of these files, a batch job command procedure can contain the PRINT commands necessary to print them.

If you want a batch job log to contain all output from the command procedure, including printed listings of compiler or linker output files, you can do either of the following:

- Use the TYPE command instead of the PRINT command in the command procedure. The TYPE command writes to SYS$OUTPUT. In a batch job, SYS$OUTPUT is equated to the batch job log file.
- Use qualifiers on appropriate commands to direct the output to SYS$OUTPUT.

  Note that if you use this technique, the output files are not saved on disk unless you save the log file.

18.5.24 Examples

- When the command procedure shown in this example completes processing, there are three separate output listings: the batch job log, the compiler listing, and the linker map:

```
$ FORTRAN/LIST BIGCOMP
$ PRINT BIGCOMP.LIS
$ LINK/MAP/FULL BIGCOMP
$ PRINT BIGCOMP.MAP
```

- The following example shows how to use qualifiers to direct the output to SYS$OUTPUT:

```
$ FORTRAN/LIST=SYS$OUTPUT  BIGCOMP
$ LINK/MAP=SYS$OUTPUT/FULL BIGCOMP
```

  When these commands are executed in a batch job, the output files from the compiler and the linker are written directly to the log file.

18.5.25 Changing Batch Job Characteristics

After a job has been submitted to the queue but before the job starts to execute, you can use the SET ENTRY or the SET QUEUE/ENTRY command with the appropriate qualifiers to change characteristics associated with the job.

18.5.26 Example

The following example shows two methods you can use to change the name of a batch job while it is pending in a batch queue:

```
$ SET QUEUE/ENTRY=209/NAME=NEW_NAME SYS$BATCH

$ SET ENTRY 209 /NAME=NEW_NAME
```

Both of these commands change the name of job number 209 to NEW_NAME.

18.5.27 Changes You Can Make to Batch Entries

The following list contains some of the changes you can make with the SET ENTRY or SET QUEUE/ENTRY commands. For a complete list of qualifiers, see the *OpenVMS DCL Dictionary*. Note that most of the qualifiers allowed with the SUBMIT command can also be used with SET ENTRY and the SET QUEUE/ENTRY commands.

You can make the following changes:

- Delay processing of a job.

  Use the /AFTER qualifier to specify a time after which the job can be executed. Use the /HOLD qualifier to hold a job until you explicitly release it.

- Release a job.

  Use the /NOHOLD or /RELEASE qualifier to release a job that was submitted with the /HOLD or /AFTER qualifiers.

- Send a job to a different queue.

  Use the /REQUEUE qualifier to change the queue on which the job will execute.

- Change execution characteristics.

  Change execution characteristics such as working set default, working set extent, working set size, job scheduling priority, and CPU time limit.

- Change the parameters to be passed to a job.

  Use the /PARAMETERS qualifier to change the parameters.

18.5.28 SUBMIT Command Qualifiers

Following are the qualifiers you can specify with the SUBMIT command. Note that you can also specify execution characteristics such as working set default, working set extent, working set size, job scheduling priority, and CPU time limit.

/AFTER
Specifies a time after which the batch job can execute. The job remains in the batch queue until the specified time. To hold a job in the queue until you explicitly release it, use the /HOLD qualifier. (To release a job that is being held, use the SET ENTRY/RELEASE command.)

/NAME
Specifies a name for the batch job. Otherwise, the job name defaults to the file name of the first (or only) command procedure in the job.

/NOTE
Specifies a message string to appear as part of the display for a SHOW QUEUE/FULL command. Allows you to convey information about the job to the operator or system manager.

/NOTIFY
Requests notification of job completion. The system sends a message to your terminal when the batch job finishes executing.

/PARAMETERS
Passes parameters to a batch job.

/NOPRINTER or /KEEP
Saves a batch job log file.

/QUEUE
Sends a batch job to a queue other than SYS$BATCH. To execute a command procedure that is located on a remote node, use the /REMOTE qualifier. This sends the job to SYS$BATCH at the remote node.

/RESTART
Enables you to restart the job if the system fails while the job is executing.

/RETAIN
Keeps a batch job in a queue after it completes. You can use the SHOW QUEUE or SHOW ENTRY commands to see the job's completion status.

18.5.29 Displaying Jobs in Batch Queues

Once a job has been entered in a batch job queue, you can monitor its status with the SHOW ENTRY command or the SHOW QUEUE command. If you have no jobs in the queue, the system displays the following message:

```
$  SHOW QUEUE BOSTON_BATCH
Batch queue BOSTON_BATCH, on BOSTON::
```

To see complete information on your jobs, use the /FULL qualifier with the SHOW ENTRY or SHOW QUEUE command. To see the status of other jobs in the queue, use the SHOW QUEUE/ALL command.

18.5.30 Examples

- In the following example, entry number 999 is diplayed:

```
$ SUBMIT EXCHAN.DAT
Job EXCHAN (queue SYS$BATCH entry 999) started on SYS$BATCH
$ SHOW ENTRY 999

  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
    999  EXCHAN          BLASS             3  Executing
         On batch queue SYS$BATCH

$ SUBMIT/NOPRINTER/PARAMETER=STATS.DAT UPDATE
Job UPDATE (queue SYS$BATCH entry 1080) started on BOSTON_BATCH
$ SHOW QUEUE BOSTON_BATCH
Batch queue BOSTON_BATCH on BOSTON::

  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
   1080  UPDATE          ODONNELL         36  Executing
```

- In the following example, the /FULL qualifier displays statistics about BOSTON_BATCH and characteristics associated with job number 999:

```
$ SHOW ENTRY/FULL 999

  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
    999  EXCHAN          BLASS             3  Executing
         On batch queue BOSTON_BATCH
         Submitted 11-DEC-1995 13:12 /PRIORITY=100
         WRKD:[BLASS]EXCHAN.DAT;3

$ SHOW QUEUE/FULL BOSTON_BATCH
Batch queue BOSTON_BATCH, on BOSTON::
    /BASE_PRIORITY=3 /JOB_LIMIT=5 /OWNER=[EXEC] /PROTECTION=(S:E,O:D,G:R,W:W)

  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
   1080  UPDATE          ODONNELL         36  Executing
    Submitted 11-DEC-1995 10:46 /KEEP /PARAM=("STATS.DAT") /NOPRINTER /PRIO=4
    _BOSTON$DQA2:[ODONNELL]TEMP.COM;1 (executing)
```

- In the following example, the SHOW QUEUE/ALL command is used to display all jobs in the BOSTON_BATCH queue:

```
$ SHOW QUEUE/ALL BOSTON_BATCH
Batch queue BOSTON_BATCH on BOSTON::

  Entry  Jobname         Username             Status
  -----  -------         --------             ------
    923  no privilege                         Executing
    939  no privilege                         Holding until 11-DEC-1995 19:00
   1080  UPDATE          ODONNELL             Executing
```

Note that, unless you are a privileged user, your information is limited to jobs submitted under your account.

18.5.31 Deleting and Stopping Batch Jobs

You can delete batch jobs before or during execution. To delete an entry that is pending or already executing in a batch queue, use the DELETE/ENTRY command. You need special privileges to delete a job that you did not submit. When a job terminates as a result of a DELETE/ENTRY command, the log file is neither printed nor deleted from your directory.

When you terminate a job using the DELETE/ENTRY command, it is handled as an abnormal termination because the operating system's normal job termination activity is preempted. As a result, the batch job log does not, for example, contain the standard logout message that summarizes job time and accounting information. Termination that results either from an explicit EXIT command or STOP command or the implicit execution of either of these commands (as the result of the current ON condition), however, is considered normal termination. The operating system performs proper rundown and accounting procedures after a normal termination.

18.5.32 Deleting a Batch Job

The following command deletes the job entry 210 in SYS$BATCH:

```
$ DELETE/ENTRY=210 SYS$BATCH
```

18.5.33 Restarting Batch Jobs

If the system fails while your batch job is executing, your job does not complete. When the system recovers and the queue is restarted, your job is aborted and the next job in the queue is executed. However, by specifying the /RESTART qualifier when you submit a batch job, you indicate that the system should reexecute your job if the system fails before the job is finished.

By default, a batch job is reexecuted beginning with the first line. See Chapter 15 and Chapter 16 for more information about symbols you can add to your command procedures to specify a different restarting point.

In addition to restarting a job after a system failure, you can also restart a job after you explicitly stop the job. To stop a job and then restart it on the same or a different queue, use the STOP/QUEUE/REQUEUE/ENTRY command.

18.5.34 Example

The command shown in this example stops job 212 on SYS$BATCH and requeues it on SYS$BATCH.

```
$ STOP/QUEUE/REQUEUE/ENTRY=212 SYS$BATCH
```

To enter this command, job 212 must have been submitted using the /RESTART qualifier to the SUBMIT command. When the batch job executes for the second time, the system uses the global symbol BATCH$RESTART to determine where to begin executing the job.

18.5.35 Synchronizing Batch Job Execution

You can use the SYNCHRONIZE and WAIT commands within a command procedure to place the procedure in a wait state. The SYNCHRONIZE command causes the procedure to wait for the completion of a specified job. The WAIT command causes the procedure to wait for a specified period of time to elapse.

If you specify a job name with the SYNCHRONIZE command, note that the jobs to be synchronized must be associated with your user name. (A job is associated with the user name of the process that submits it.) To synchronize jobs for different users, you must use the /ENTRY qualifier with the SYNCHRONIZE command to specify the job entry number.

18.5.36 Examples

- In the following example, if two jobs are submitted concurrently to perform cooperative functions, one job can contain the following command:

  ```
  $ SYNCHRONIZE BATCH25
  ```

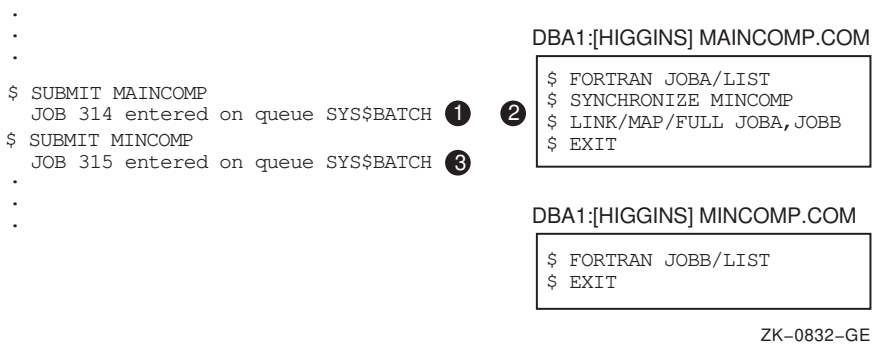
  After this command is executed, the command procedure cannot continue execution until the job identified by the job name BATCH25 completes execution.

- This SYNCHRONIZE command places the current command procedure in a wait state until job 454 completes:

  ```
  $ SYNCHRONIZE/ENTRY=454
  ```

- Figure 18–1 is an example of command procedures that are submitted for concurrent execution but must be synchronized for proper execution. Each procedure compiles a large source program.

Figure 18–1 Synchronizing Batch Job Execution



As you examine the example, note the following:

❶ Individual SUBMIT commands are required to submit two separate jobs. The first process is created.

❷ After the FORTRAN command is executed, the SYNCHRONIZE command is executed. If job 315 is either current or pending, job 314 will not execute the next command.

❸ If job 315 has completed execution, job 314 continues with the next command.

18.5.37 Using the WAIT Command

The WAIT command is useful for command procedures that must have access to a shared system resource such as a disk or tape drive.

18.5.38 Example: Using the WAIT Command

other window
The following example shows a procedure that requests the allocation of a tape drive:

```
$ TRY:
$       ALLOCATE DM: RK:
$       IF $STATUS THEN GOTO OKAY
$       WAIT 00:05
$       GOTO TRY
$ OKAY:
$ REQUEST/REPLY/TO=DISKS -
      "Please mount BACK_UP_GMB on ''F$TRNLNM("RK")'"
    .
    .
    .
```

If the WAIT command does not complete successfully, the procedure places itself in a wait state. After a 5-minute interval, it retries the request.

The IF command following the ALLOCATE request checks the value of $STATUS. If the value of $STATUS indicates successful completion, the command procedure continues. Otherwise, the procedure executes the WAIT command; the WAIT command specifies a time interval of five minutes. After waiting five minutes, the next command, GOTO, is executed and the request is repeated. This procedure continues looping and attempting to allocate a device until it succeeds or until the batch job is deleted or stopped.

19

Security: Controlling Access to Resources

19.1 Overview

This chapter describes some of the ways OpenVMS protects and audits your system resources. It includes information on:

- Displaying the rights identifiers of your process
- Security profile of objects
- Interpreting protection codes
- Default file protection
- Accessing files across networks
- Auditing access to your account and files

19.1.1 Resources

Refer to the following for additional security information:

- For information on protecting objects and system security, in general, see the *OpenVMS Guide to System Security*
- For information on commands discussed in this chapter, refer to online Help or the *OpenVMS DCL Dictionary*

19.1.2 Security Features

Some ways you can familiarize yourself with OpenVMS security features are:

- Know the rights **identifiers** associated with your process—Rights identifiers determine what resources you can access. If your process does not have the appropriate identifiers, you may be unable to access certain protected objects.

  See Section 19.2 for information on displaying your rights identifiers.
- Display security profiles of protected objects—A security profile contains information about protected objects. You can change the security profile of objects that you own to make them accessible or inaccessible to other users.

  See Section 19.3 for information on security profiles.
- Know how to access files across networks—This can be accomplished by using access control strings or proxy login accounts.

  See Section 19.6 for information on accessing remote files.

- Audit access to your account and files—This can be accomplished by closely observing any login messages and by working in conjunction with your system manager to audit your files.

  See Section 19.7 for information on auditing access to your account and files.

19.2 Displaying the Rights Identifiers of Your Process

19.2.1 Overview

All processes that attempt to access protected objects carry credentials known as rights identifiers. All protected objects list a set of access requirements that specify who has a right to access the object in a given manner. If an accessing process' rights identifiers do not match those of the object, access is denied.

19.2.2 Example: Displaying Rights Identifiers

The following example shows how to display the identifiers for your current process using the SHOW PROCESS command:

```
$ SHOW PROCESS/ALL
25-JUN-1991 15:23:18.08   User: GREG            Process ID:   34200094
                          Node: ACCOUNTS        Process name: "_TWA2:"

Terminal:           TWA2:
User Identifier:    [DOC,GREG]    (1)
Base priority:      4
Default file spec:  WORK1:[GREG.FISCAL_91]

Devices allocated:  ACCOUNTS$TWA2:

Process Quotas:
   .
   .
   .
Process rights:
 INTERACTIVE    (2)
 LOCAL          (3)
 SALES          (4)
 MINDCRIME                          resource  (5)

System rights:
 SYS$NODE_ACCOUNTS    (6)
```

There are three types of rights identifiers: UIC, environmental, and general. Output from the SHOW PROCESS command displays all three:

(1) UIC identifier, indicating user Greg is a member of the DOC group

(2) Environmental identifier, indicating user Greg is an interactive user

(3) Environmental identifier, indicating user Greg is logged in locally

(4) General identifier, indicating user Greg is also a member of the SALES group

❺ General identifier, indicating Greg holds the MINDCRIME identifier with the resource attribute so he can charge disk space to the identifier

❻ Environmental identifier, indicating user Greg is working from the ACCOUNTS node

19.3 Security Profile of Objects

19.3.1 Overview

Because the operating system supports many users simultaneously, it has built-in security mechanisms to prevent one user's activities from interfering with another's. Protection codes, access controls, and hardware design together protect the use of memory, shareable devices, and data so many users can share the system. An object's security profile is comprised of the user identification code (UIC), the ACL, and the protection codes assigned to that object. You can display or modify the security profile of any object that you own.

19.3.2 Displaying a Security Profile

To see the security profile of any protected object, use the DCL command SHOW SECURITY. For example, the following command requests security information about the file 95_FORECAST.TXT:

```
$ SHOW SECURITY 95_FORECAST.TXT

WORK_DISK$:[GREG]95_FORECAST.TXT;1 object of class FILE
      Owner: [ACCOUNTING,GREG]
      Protection: (System: RWED, Owner: RWED, Group: RE, World)
      Access Control List: <empty>
```

The display indicates the file 95_FORECAST.TXT is owned by user Greg. It also lists the file's protection code, which gives read, write, execute, and delete access to system users and to the owner. The code grants read and execute access to group users and provides no access to world users. (See Section 19.4 for further explanation.) There is no ACL on the file.

19.3.3 Modifying a Security Profile

You can provide new values for the owner, protection code, or ACL of a protected object, or you can copy a profile from one object to another by using the SET SECURITY command.

For example, the SHOW SECURITY display in Section 19.3.2 shows the file 95_FORECAST.TXT is owned by user Greg. As owner, he can change the protection code for that file. Originally, the code gave no access to users in the world category. Now, Greg has changed that to allow read and write access to world users:

```
$ SET SECURITY/PROTECTION=(W:RW) 95_FORECAST.TXT
```

The SHOW SECURITY command verifies the new protection code for the file:

```
$ SHOW SECURITY 95_FORECAST.TXT

95_FORECAST.TXT object of class FILE

     Owner: [GREG]
     Protection: (System: RWED, Owner: RWED, Group: RE, World: RW)
     Access Control List: <empty>
```

19.4 Interpreting Protection Codes

19.4.1 Protection Code Format

A protection code controls the type of access allowed (or denied) to a particular user or group of users. It has the following format:

[category: list of access allowed (, category: list of access allowed,...)]

19.4.2 Production Categories

Categories include system (S), owner (O), group (G), and world (W). Each category can be abbreviated to its first character. Categories have the following definitions:

System	Any user process or application whose UIC is in the range 1 through 10 (octal), has SYSPRV privilege, or is in the same group as the owner and holds GRPPRV.
Owner	Any user process or application whose UIC is identical to the UIC of the object.
Group	Any user process or application whose group UIC is identical to the group UIC of the object.
World	Any user process or application on the system.

When specifying more than one user category, separate the categories with commas and enclose the entire code in parentheses. You can specify user categories and access types in any order.

A null access specification means no access, so when you omit an access type for a user category, that category of user is denied that type of access. To deny all access to a user category, specify the user category without any access types. Omit the colon after the user category when you are denying access to a category of users.)

19.4.3 Access List

For files, an access list includes read (R), write (W), execute (E), or delete (D) access types. The access type is assigned to each ownership category and is separated from its access types with a colon (:). File access types have the following meanings:

Read	Gives you the right to read, print, or copy a disk file. With directory files, read access gives you the right to read or list a file and use a file name with wildcard characters to look up files. Read access implies execute access.
Write	Gives you the right to write to or change the contents of a file, but not delete it. Write access allows modification of the file characteristics that describe the contents of the file. With directory files, write access gives you the right to insert or delete an entry in the catalog of files.
Execute	Gives you the right to execute a file that contains an executable program image or DCL command procedure. With a directory file, execute access gives you the right to look up files whose names you know.
Delete	Gives you the right to delete the file. To delete a file, you must have delete access to the file and write access to the directory that contains the file.

19.5 Default File Protection

19.5.1 Overview

A new file receives the default UIC-based protection and the default access control list (ACL) of its parent directory. An ACL is a collection of entries that define the access rights a user or group of users has to a particular protected object such as file, directory, or device.

You can use either default UIC protection or default ACL protection to override the default UIC-based protection given to new files.

19.5.2 Default UIC protection

The operating system provides each process with the following UIC-based protection:

```
(S:RWED, O:RWED, G:RE, W)
```

By default, users with a system UIC and the owners of objects have full access to the object, users in the same UIC group as the object owner have read and execute access to the object, and all other users are denied access to the object. To change the default protection for files that you create, enter the SET PROTECTION command with the /DEFAULT qualifier. For example, if you enter the following command in your login command procedure, you grant all processes read and execute access to any files that you create. (Remember that you must execute the login command procedure for this command to execute.)

```
$ SET PROTECTION = (S:RWED,O:RWED,G:RE,W:RE)/DEFAULT
```

19.5.3 Default ACL protection

You can override default UIC protection for specified directories or subdirectories by placing a default protection ACE in the ACL of the appropriate directory file. The default protection specified in the ACE is applied to any new file created in the specified directory or subdirectory of the directory. The following ACE, which must be in the ACL of a directory file, specifies that the default protection for that directory and the directory's subdirectories allow system and owner processes full access, group processes read and execute access, and world users no access.

```
$ SET SECURITY/ACL = (DEFAULT_PROTECTION,S:RWED,O:RWED,G:RE,W:)
[JONES]PERSONAL.DIR
```

To specify a default identifier ACE to be copied to the ACL of any file subsequently created in the directory, specify the DEFAULT option in the directory file's identifier ACL.

19.5.4 Example

The ACE shown in the following example is applied to a directory file and denies network users access to all files created in the directory:

```
$ SET SECURITY/ACL = (IDENTIFIER=NETWORK,OPTIONS=DEFAULT,ACCESS=NONE) -
_$ [JONES]PERSONAL.DIR
```

19.5.5 Renaming Files

A renamed file's protection is unchanged. A new version of an existing file receives the UIC-based protection and ACL of the previous version. (Use the /PROTECTION qualifier of the BACKUP, COPY, CREATE, and SET FILE commands to override the default UIC-based protection.)

19.5.6 Explicit File Protection

You can explicitly specify UIC-based protection for a new file with the /PROTECTION qualifier (valid with the BACKUP, COPY, and CREATE commands).

You can change the UIC-based protection on an existing file with the SET SECURITY/PROTECTION command.

After a file is created and you have created an ACL for the file, you can modify the ACL and add as many entries to it as you want. The protection specified by the ACL overrides the file's user identification code protection.

19.5.7 Examples

- In the following example, UIC-based protection is specified:

  ```
  $ CREATE MAST12.TXT/PROTECTION=(S:RWED,O:RWED,G,W)
  ```

- In the following example, the UIC-based protection is changed on the file MAST12.TXT:

  ```
  $ SET SECURITY/PROTECTION=(S:RWED,O:RWED,G:RE,W) MAST12.TXT
  ```

19.6 Accessing Files Across Networks

19.6.1 Access Control Strings

You can include network access control strings in the file specifications of DCL commands that perform operations across the DECnet for OpenVMS network. The access control strings permit a user on a local node to access a file on a remote node.

19.6.2 Access Control String Format

An access control string consists of the user name for the remote account and the user's password enclosed within quotation marks, as follows:

```
NODE"username password"::disk:[directory]filename.filetype
```

19.6.3 Caution: ACLs and Security

Because access control strings include sufficient information to allow someone to break in to the remote account, they create serious security exposure.

19.6.4 Protecting Access Control Strings

To protect access control string information, do the following:

- Avoid revealing the information on either hardcopy or video terminals. If you use a hardcopy terminal, dispose of the output properly. If you use a video terminal, clear the screen and empty the recall buffer with the DCL command RECALL /ERASE when the network job is completed. This prevents another user from seeing the password, either by displaying the command line with the Ctrl/B sequence or with the DCL command RECALL/ALL.
- Do not place networking commands that include access control strings in command procedures where they would be likely targets for discovery.
- If you must put access control strings in your command procedures, provide these files with optimum file protection.

19.6.5 Using Proxy Login Accounts to Protect Passwords

To avoid the need for access control strings, you might prefer to use proxy login accounts. Proxy logins let you access files across a network without specifying a user name or password in an access control string. Thus, proxy logins have the following security benefits:

- Passwords are not echoed on the terminal where the request originates.
- Passwords are not passed between systems where they might be intercepted in unencrypted form.
- Passwords are not needed in command files to perform the remote access steps.

19.6.6 Initiating Proxy Logins

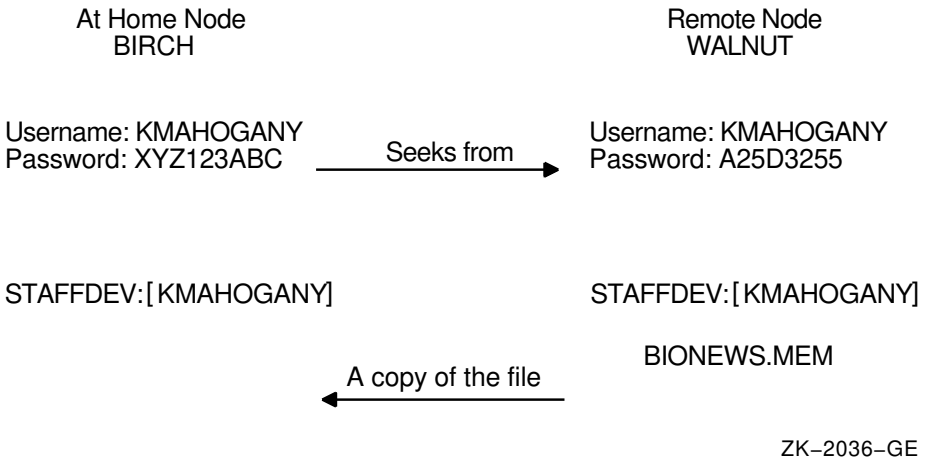
Before you can initiate a proxy login, the system or security administrator at the remote node must create a proxy account for you. Proxy accounts, like regular accounts, are created with the OpenVMS Authorize utility (AUTHORIZE). They are usually nonprivileged accounts. Security administrators can allow you access to one default proxy account and up to 15 other proxy accounts. While proxy logins require more setup effort on the part of system managers, they provide more secure network access and eliminate the need for users to enter access control strings.

19.6.7 Examples

The following example illustrates the differences between a normal network login request and a proxy login request. For each example, the following conditions exist:

- The user KMAHOGANY has two user accounts:
  - An account on node BIRCH with the password ''XYZ123ABC''
  - An account on node WALNUT with the password ''A25D3255''
- KMAHOGANY has logged in to node BIRCH.
- KMAHOGANY wants to copy the file BIONEWS.MEM from the default device and directory of the account on the node WALNUT.

The following diagram illustrates these conditions:



- The user KMAHOGANY could use an access control string to copy the file BIONEWS.MEM, as follows:

```
$ COPY WALNUT"KMAHOGANY A25D3255"::BIONEWS.MEM  BIONEWS.MEM
```

  Notice that the password A25D3255 echoes. Anyone who observes the screen can see it.

- If KMAHOGANY has proxy access from node BIRCH to the account on node WALNUT, the command for copying the file BIONEWS.MEM is as follows:

  ```
  $ COPY WALNUT::BIONEWS.MEM   BIONEWS.MEM
  ```

  KMAHOGANY does not need to specify a password in an access control string. Instead, the system performs a proxy login from her account on node BIRCH into her account on node WALNUT. There is no exchange of passwords.

19.6.8 General Access Proxy Accounts

Your security administrator can also authorize groups of users from foreign nodes to share in the use of a general access proxy account. For example, the security administrator at node WALNUT can create a general access account with the following conditions:

- The user name GENACCESS.
- Access limited to network logins.
- A password known only to the owner of the account. (None of the remote users need to know it.) This helps to protect the account.
- The default device and directory STAFFDEV:[BIOSTAFF].

If the security administrator grants BIRCH::KMAHOGANY proxy access to the GENACCESS account, the user KMAHOGANY can copy the file BIONEWS.MEM by entering the following command:

```
$ COPY WALNUT::[KMAHOGANY]BIONEWS.MEM   BIONEWS.MEM
```

Note that KMAHOGANY must specify the directory [KMAHOGANY] because the file BIONEWS.MEM is not in the default device and directory for the GENACCESS account (STAFFDEV:[BIOSTAFF]). In addition, the protection for the file BIONEWS.MEM must permit access to the GENACCESS account. Otherwise, the command fails.

19.6.9 Specify Proxy Accounts

If you have access to more than one proxy account on a given node and you do not want to use the default proxy account, specify the name of the proxy account. For example, to use a proxy account called PROXY2 instead of the GENACCESS account (the default), KMAHOGANY enters the following command:

```
$ COPY WALNUT"PROXY2"::[KMAHOGANY]BIONEWS.MEM BIONEWS.MEM
```

This command uses the PROXY2 account to copy the file BIONEWS.MEM from the [KMAHOGANY] directory on node WALNUT.

19.7 Auditing Access to Your Account and Files

19.7.1 Overview

Although it is the security administrator's job to monitor the system for possible break-in attempts, you can assist the security administrator in auditing access to your account and files.

19.7.2 Observing Your Last Login Time

The OpenVMS system maintains information in your UAF record about the last time you logged in to your account. Your security administrator decides whether the system should display this information at login time. Sites with medium to high security requirements frequently display this information and ask users to check it for unusual or unexplained successful logins and unexplained failed logins.

If there is a report of an interactive or a noninteractive login at a time when you were not logged in, report it promptly to your security administrator. Also change your password. The security administrator can investigate further by using accounting files and audit logs.

If you receive a login failure message and cannot account for the failure, it is likely that someone has been trying to access your account unsuccessfully. Check your password to ensure that it adheres to all recommendations for password security described in Section 2.9. If not, change your password immediately.

If you expect to see a login failure message and it does not appear or if the count of failures is too low, change your password. Report either of these indications of login failure problems to your security administrator.

19.7.3 Asking Your Security Administrator to Enable Auditing

The security administrator can select one or more types of events that warrant special attention when they occur. When such an event is detected, the security administrator directs the system to send an audit to the system security audit log file or an alarm to terminals enabled as security operator terminals. For example, the security administrator might identify one or more files for which write access is prohibited. An audit can be enabled or an alarm can be set to indicate attempted access to these files.

If you suspect a break-in to your account, change your password. You might want to request that your security administrator implement auditing on sensitive files.

19.7.4 Events That Can Trigger Security Alarms

Events triggering an audit or alarm can include the following:

Example of Events Initiating Security Audits or Alarms	
Installation of images	Modifications to: • System and user passwords • System authorization file • Network proxy file • Rights database
Certain types of file access	
Volume mounts and dismounts	
Access event requested by an ACL file or global section	Logins, logouts, login failures, break-in attempts

19.7.5 Example: Unauthorized Access to an Audited File

In the following example, assume you decide to audit the file CONFIDREVIEW.MEM. If user ABADGUY accesses CONFIDREVIEW.MEM and has delete access, the following audit record is written to the system security audit log file:

```
%%%%%%%%%%  OPCOM  11-DEC-1995 09:21:11.10  %%%%%%%%%%
Message from user AUDIT$SERVER on BOSTON
Security audit (SECURITY) on BOSTON, system id: 19424
Auditable event:        Attempted file access
Event time:              11-DEC-1995 09:21:10.84
PID:                    23E00231
Username:               ABADGUY
Image name:             BOSTON$DUA0:[SYS0.SYSCOMMON.][SYSEXE]DELETE.EXE
Object name:            _BOSTON$DUA1:[RWOODS]CONFIDREVIEW.MEM;1
Object type:            file
Access requested:       DELETE
Status:                 %SYSTEM-S-NORMAL, normal successful completion
Privileges used:        SYSPRV
```

The auditing message reveals the name of the perpetrator, the method of access (successful deletion accomplished by using the program [SYSEXE]DELETE.EXE), time of access (9:21 A.M.), and the use of a privilege (SYSPRV) to gain access to the file. With this information, the security administrator can take action.

19.7.6 Security Audit Log Files

Security audit messages are written to the security audit log file every time any file is accessed and meets the conditions specified in the audit entry of the ACL for that file (see Section 19.7.7). Access to the file CONFIDREVIEW.MEM, as well as access to any file on the system that is protected with security auditing, prompts an audit record to be written to the security audit log file.

After auditing has been introduced, check with your security administrator periodically to see if any additional break-ins have occurred.

19.7.7 Adding ACEs to Sensitive Files

If you have key files that might have been accessed improperly, you might want to develop a strategy with your security administrator to audit access to the files.

Once you review the situation and ensure that you have done everything possible to protect your files with standard protection codes and general ACLs (described in the *OpenVMS Guide to System Security*), you may conclude that security auditing is required.

To specify security auditing, you can add special access control entries (ACEs) to files you own or to which you have control access. Keep in mind, however, that the audit log file is a systemwide mechanism, so Digital recommends that a site security administrator control the use of file auditing. Although you can add auditing ACEs to files over which you have control, the security administrator has to enable auditing of files on a system level.

If you suspect break-in attempts to your account, the security administrator may temporarily enable auditing for all file access. The security administrator can also enable auditing to monitor read access to your files to catch file browsers.

An access violation of one file frequently indicates access problems with other files. Therefore, the security administrator may need to monitor access to all key files having security-auditing ACEs. When undesired access is gained to key files, the security administrator must take immediate action.

19.7.8 Example

In the following example, user RWOODS and his security administrator concur that they must know when a highly confidential file, CONFIDREVIEW.MEM, is being accessed, so RWOODS adds an entry to the existing ACL for the file CONFIDREVIEW.MEM:

```
$ SET SECURITY/ACL=(ALARM=SECURITY,ACCESS=READ+WRITE-
_$ +DELETE+CONTROL+FAILURE+SUCCESS) CONFIDREVIEW.MEM
```

A
Customizing EVE

A.1 Overview

This appendix describes how to use startup files to modify the standard EVE editor. It includes information on:

- Defining EVE keys
- Learn sequences
- Setting and saving attributes
- Using DECTPU within EVE
- Using DECTPU procedures to extend EVE
- Creating section files
- Creating command files
- Creating initialization files
- Saving your customizations in startup files
- EVE commands for saving attributes
- Saving attributes
- Key definitions in EVE startup files
- Converting from EDT to EVE

A.1.1 References

For additional information, refer to:

- The *OpenVMS DCL Dictionary* or online Help for DCL command information
- The *DEC Text Processing Utility Reference Manual* for information on the DECTPU debugging package
- The *Guide to the Extensible Versatile Editor* for information about customizing EVE

A.1.2 About Startup Files

Startup files hold key definitions and editing commands that set the characteristics of the editing environment. Startup files can also hold DECTPU procedures, which augment the editing capability of the standard EVE editor.

By placing your definitions and procedures in a startup file, you can invoke the editor and automatically establish the editing environment your task requires.

A.1.3 Tailoring the Standard Editor

EVE provides the following ways of tailoring the standard editor to meet your editing requirements:

- Defining editing keys

  You can assign an editing command to a key so that commands are faster to enter. For example, you can bind the EVE command ERASE WORD to the key sequence Ctrl/D.

- Creating learn sequences

  You can assign a series of commands or keystrokes to one key. For example, you can create a learn sequence where you can press one key to insert a new phone number into a standard memo heading.

- Using the DEC Text Processing Utility (DECTPU) to write editing procedures

  Because EVE is built on a powerful, programmable utility called the DEC Text Processing Utility, you can expand the editor beyond its standard set of commands by using DECTPU procedures. With DECTPU language statements, you can write procedures that create features that are not available in standard EVE. For example, you can write a procedure to transpose two characters and assign this procedure to a key.

A.2 Defining EVE Keys

A.2.1 Overview

You can define keys to execute EVE commands or to enter a series of keystrokes, called a learn sequence.

A.2.2 Undefinable Keys

EVE does not let you define the Return key (Ctrl/M), the space bar, or any printing characters (such as letters, digits, and punctuation marks) on the main keyboard.

In addition, Digital recommends that you do not define the following keys and control key sequences (some cannot be defined unless you use special terminal settings):

Break
Delete or
Escape or Ctrl/[
F1 to F6 (VT200 and VT300 series terminals)
Help (PF2 on VT100 series terminals)
Noscroll
Shift

Ctrl/C
Ctrl/I (Tab key)
Ctrl/O
Ctrl/Q
Ctrl/R (which EVE defines as REMEMBER)
Ctrl/S
Ctrl/T
Ctrl/U (which EVE defines as START OF LINE)
Ctrl/X
Ctrl/Y

You can define all other keys (including control keys) except those noted. You can redefine the Do key, as long as you have assigned the DO command to another key.

A.2.3 Obsolete Keys

The SET SHIFT KEY and SET NOSHIFT KEY commands are obsolete. Instead, use the SET GOLD KEY and SET NOGOLD KEY commands, respectively.

A.2.4 Defining Keys to Execute EVE Commands

By defining keys, you can create editing keys to enter EVE commands you use frequently. You can define a key to execute an EVE command by using the DEFINE KEY command or by using an initialization file. If you are using help and press a key to which you have assigned an EVE command, EVE provides the help text for that command. Key definitions are discarded when you end an EVE editing session, unless you use the SAVE ATTRIBUTES command or SAVE EXTENDED EVE command to save key definitions from one editing session to the next.

A.2.5 Using the DEFINE KEY Command

The DEFINE KEY command assigns an EVE command to a single key, a GOLD key combination, or control key sequence. You can enter the DEFINE KEY command, the key to be defined, and the command on a single command line, or you can enter the DEFINE KEY command and let EVE prompt you.

To enter the DEFINE KEY command on a single command line, use the following command syntax:

DEFINE KEY [=key-name] command

The elements are as follows:

key-name The key to be defined

command The command you assign to the key

For example, the following command assigns the MOVE BY WORD command to keypad key 1 (KP1):

```
Command: DEFINE KEY=KP1 MOVE BY WORD
```

The following command assigns the FILL command to Ctrl/F:

```
Command: DEFINE KEY=Ctrl/F FILL
```

You can use one of three different separators when specifying key names: an underscore (_), a dash (–), or a slash (/). For example, the Ctrl/F key can be specified as Ctrl_F, Ctrl–F, or Ctrl/F.

To use the DEFINE KEY command and let EVE prompt you, invoke EVE, then press the Do key and enter the DEFINE KEY command, as follows:

```
[End of file]

Buffer: MAIN                              | Write | Insert | Forward
Command: DEFINE KEY
```

Type the EVE command you want to assign to a key and press the Return key.

```
[End of file]

Buffer: MAIN                              | Write | Insert | Forward
EVE command: START OF LINE
```

Press the key to be associated with the EVE command.

```
[End of file]

Buffer: MAIN                              | Write | Insert | Forward
Press the key you want to define: F20
```

The message ‘‘Key defined’’ appears if you have successfully defined a key.

Another way to assign EVE commands to keys is to create an initialization file. You can define keys and set the characteristics of an editing session in the initialization file. The file contains EVE commands and key definitions, and is executed when you invoke EVE. Use the syntax given in this section to put DEFINE KEY commands in the file. For more information about initialization files, see the online help topic called Initialization Files.

To remove a key definition, use the UNDEFINE KEY command.

Section A.2.12 contains more examples of defining keys to execute EVE commands.

You can type the name of a key as a parameter for the DEFINE KEY, SET GOLD KEY, SHOW KEY, and UNDEFINE KEY commands. EVE key names are generally the same as the labels on the keys—you can specify them by their labels as well as by their positional number on one of the keypads. For example, the 7 on the numeric keypad is named KP7 and the keys on the minikeypad are named E1 to E6.

A.2.6 Key Name Abbreviations

You can abbreviate key names as long as your abbreviation is not ambiguous. For example, G Rem is a valid abbreviation for GOLD Remove and G R is an abbreviation for GOLD R. The case of letters does not matter in a key definition.

A.2.7 Defining Control Keys

You can specify control keys by using Ctrl, Control, or the circumflex character (^). For example, Ctrl/A, Control/A, and ^A are the same. For a list of the control keys defined by EVE, see the EVE online help topic called Control Keys.

In specifying control keys or GOLD key sequences, use a dash (–), slash (/), or underscore (_) as a delimiter in the key name (for example, GOLD-F20, Alt/A, or Ctrl_N.) Thus, in an initialization file, you can use commands with typed key names such as the following:

```
DEFINE KEY= Ctrl/P  MOVE BY PAGE
DEFINE KEY= GOLD-N  NEXT BUFFER
DEFINE KEY= KP7     CENTER LINE
SET GOLD KEY F17
```

A.2.8 Differences Between EVE and DECTPU

Some EVE key names are different from key names that you use in DECTPU command files, as shown in Section A.2.8.

In EVE Commands	In DECTPU Procedures
GOLD A	KEY_NAME (A, SHIFT_KEY)
GOLD MINUS	KEY_NAME (MINUS, SHIFT_KEY)
Ctrl/D	Ctrl_D KEY
SHIFT/F14	KEY_NAME (F14, SHIFT_MODIFIED)

A.2.9 EVE Key Names

The following table lists EVE key names and the key labels on the keyboard or keypads. Some keys may not appear on some terminals. (For example, VT100 series terminals do not have the F1 to F20 keys. VT200 , VT300, and VT400 series terminals do not have Backspace and line-feed keys.) Do not use these key names in DECTPU built-in procedures. See the table ''Keywords Used for Key Names'' in the *Guide to the DEC Text Processing Utility* for the correct keywords to use in DECTPU built-in procedures.)

Key	Key Name
F7 . . . F20	F7 . . . F20
Help	HELP or F15
Do	DO or F16
Find	FIND or E1
Insert Here	INSERT_HERE or E2
Remove	REMOVE or E3
Select	SELECT or E4

Key	Key Name
Prev Screen	PREV_SCREEN or E5
Next Screen	NEXT_SCREEN or E6
↑ (up arrow)	UP
← (left arrow)	LEFT
↓ (down arrow)	DOWN
→ (right arrow)	RIGHT
PF1 . . . PF4	PF1 . . . PF4
0 . . . 9 (numeric keypad)	KP0 . . . KP9
– (numeric keypad)	MINUS
. (numeric keypad)	PERIOD
, (numeric keypad)	COMMA
<⌫ or Delete	DELETE
Tab or TAB	TAB or Ctrl/I
Backspace	BS or Ctrl/H
Line-feed	LF or Ctrl/J

A.2.10 Undefinable Keys

You cannot define any of the following keys:

F1 to F6
Compose Character
Ctrl (by itself)
Return or Ctrl–M
Break
Escape or Ctrl/[
Lock or Caps Lock
No Scroll
Set-up
Shift

In addition, EVE does not let you define typing keys on the main keyboard (except in combination with a modifier), a key defined as DO if it is the only key defined as DO or the key currently set as GOLD, if any.

A.2.11 Keys That You Should Not Define

Digital recommends that you do not define the following keys and control keys. You can define these control keys, but you cannot execute them unless you set terminal characteristics in special ways.

- Delete or <⌫ (which EVE defines as DELETE)
- Help or on VT100 terminals, PF2
- Ctrl/B (which EVE defines as RECALL)

- Ctrl/C
- Ctrl/O
- Ctrl/Q
- Ctrl/R (which EVE defines as REMEMBER to end a learn sequence)
- Ctrl/S
- Ctrl/T
- Ctrl/U (which EVE defines as ERASE START OF LINE)
- Ctrl/V (which EVE defines as QUOTE)
- Ctrl/X
- Ctrl/Y

If you redefine Ctrl/B or Ctrl/R, you should define other keys as RECALL and REMEMBER, respectively, because you can execute those commands only by pressing a key.

A.2.12 Defining the GOLD Key

You can assign two definitions to the same editing key if you create a GOLD key. Invoke one key definition by pressing the editing key. Invoke the other key definition by first pressing the GOLD key and then pressing the editing key.

To define a GOLD key, enter the SET GOLD KEY command and press the key you want to use as the GOLD key. When you successfully define the key, the message ''GOLD key set.'' appears in the Messages buffer. EVE does not have a default GOLD key.

A.2.13 Using the GOLD Key

After you create a GOLD key, you can use the GOLD key definitions supplied by EVE. To see a diagram of these key definitions, enter the command HELP KEYPAD. The GOLD key definitions appear in the display in reverse video.

If you press the GOLD key by mistake, press the Select key to cancel it. Use the SAVE ATTRIBUTES command or the SAVE EXTENDED EVE command to save key definitions from one editing session to the next.

A.2.14 GOLD Key Combinations

The following table lists the GOLD key combinations on the EVE keypad and the definitions associated with them:

Key	Definition
GOLD F13	Restore Word (except with the WPS keypad)
GOLD Help	Help keys
GOLD Find	Wildcard Find

Key	Definition
GOLD Insert Here	Restore
GOLD Remove	Store Text
GOLD Select	Reset
GOLD Prev Screen	Previous Window
GOLD Next Screen	Next Window
GOLD ↑	Top
GOLD ↓	Bottom
GOLD ←	Start of Line
GOLD →	End of Line

A.2.15 Tutorial: Creating GOLD Key Definitions

You can also use the GOLD key to create your own key definitions. This tutorial demonstrates how to define a GOLD key and assign two commands to a single key. The tutorial defines the 4 key on the numeric keypad as the GOLD key and then assigns the BOTTOM and TOP commands to the Ctrl/G key. Thus, pressing Ctrl/G alone enters the BOTTOM command and pressing the GOLD key followed by Ctrl/G enters the TOP command.

To define a GOLD key and the bottom and top keys, follow these steps:

Step	Task
1	Define the GOLD key: a. Press the Do key, type SET GOLD KEY, and press the Return key. b. Press the 4 key on the numeric keypad.
2	Define the bottom key: a. Press the Do key, type DEFINE KEY, and press the Return key. b. Type BOTTOM and press the Return key. c. Press Ctrl/G. Ctrl/G is now defined as BOTTOM.
3	Define the GOLD Ctrl/G key as the top key: a. Press the Do key, type DEFINE KEY, and press the Return key. b. Type TOP and press the Return key. c. Press and hold down the GOLD key (4 on the numeric keypad) and press Ctrl/G.

For the rest of your editing session, when you press Ctrl/G, EVE executes the BOTTOM command. When you press the GOLD key (4 on the numeric keypad) followed by Ctrl/G, EVE executes the TOP command.

A.2.16 Removing GOLD Keys

You cannot define more than one GOLD key at a time. To remove a GOLD key definition, enter the SET NOGOLD KEY command, then press the key you want to undefine. You can also define another GOLD key, which removes the original GOLD key.

A.2.17 Defining GOLD Keys in Initialization Files

Use the following format to define a GOLD key in an initialization file:

SET GOLD KEY keyname

For example, the following command defines the PF1 key as the GOLD key:

```
SET GOLD KEY PF1
```

A.3 Learn Sequences

A.3.1 Overview

The LEARN command assigns a sequence of keystrokes, called a learn sequence, to a single key or control key sequence. With learn sequences, you can enter the same series of keystrokes in a buffer any number of times by pressing one key. All learn sequences are discarded when you terminate an EVE editing session unless you use the SAVE ATTRIBUTES command or the SAVE EXTENDED EVE command to save them from one editing session to the next.

A.3.2 Defining Learn Sequences

To define a learn sequence, take these steps:

Step	Task
1	Enter the LEARN command.
2	Type the keystrokes to be remembered. You can press keys already defined, type text, or both.
3	Press Ctrl/R.
4	At the prompt, press the key to be associated with the learn sequence, such as F17 or PF3.
5	To cancel the learn sequence, press the Return key or Ctrl/M.

The message ‘‘Key sequence remembered’’ appears if you have successfully defined a key.

A.3.3 Tutorial: Defining Learn Sequences

To define a learn sequence that inserts a string of text into your file when you press Ctrl/F, follow these steps:

Step	Task

1	Invoke EVE to edit the file RHYMES.DAT.
	`She rhymes with tree,`
	`also with bee,`
	`and this one makes three.`
	`[End of file]`
	
	`Buffer: RHYMES.DAT | Write | Insert | Forward`
	
	`3 lines read from file WORKDISK:[USER]RHYMES.DAT`

2	Move the cursor to the end of the buffer. To begin the definition of the learn sequence, press the Do key, and enter the LEARN command.
3	Insert the following text, which EVE is to remember, at the end of your file: `And what is a rhyme?`
4	Press Ctrl/R.
5	Press Ctrl/F, the key to which you are assigning the learn sequence.

For the rest of the editing session, when you press Ctrl/F, EVE inserts the text ‘‘And what is a rhyme?’’ wherever the cursor is positioned at the time.

A.4 Setting and Saving Attributes

A.4.1 Overview

You can save most global attributes in a section file or DECTPU command file for future editing sessions. You can also set a default section file or command file to be created or updated for saving attributes.

A.4.2 EVE Default Settings

This list shows the EVE default settings—the settings EVE uses unless you specify otherwise. You may want to refer to this table to check which settings you want to change when creating an initialization file. Some settings are global (applying to all buffers you edit), and others are buffer specific. For example, the type of cursor motion (bound or free) and tab mode (insert, spaces, or movement) are the same for all buffers you edit, whereas you can set margins, paragraph indents, and tab stops differently for each buffer.

- **SET BOX NOSELECT**
  Disables box-style selecting, cutting, and pasting so you can select and edit standard linear ranges.

- **SET BOX PAD**
  Enables padding and overstriking for box editing, regardless of the mode of the buffer.
- **SET CURSOR FREE**
  You can move the cursor anywhere in the buffer and enter text there, as opposed to a bound cursor, which cannot move into the unused portion of the buffer. Using SET KEYPAD WPS automatically enables a bound cursor.
- **SET EXIT ATTRIBUTE CHECK**
  If you changed attributes, then when you exit or quit, EVE asks whether you want to save them.
- **SET FIND CASE NOEXACT**
  EVE finds any occurrence of a text string if you enter the search string in all lowercase.
- **SET FIND NOWHITESPACE**
  FIND and WILDCARD FIND commands match spaces and tabs in the search string exactly as entered and do not search across a line break.
- **SET FUNCTION KEYS NODECWINDOWS**
  Keeps the normal key definitions (EVE default, EDT keypad, or WPS keypad) rather than defining some keys for DECwindows functions.
- **SET KEYPAD NUMERIC**
  or
  SET KEYPAD VT100
  On VT200, VT300, and VT400 series terminals, keys on the numeric keypad are undefined, except for the PF4 and Enter keys. On VT100 series terminals, the numeric keypad is used for the EVE default key bindings. Control keys are defined the same on either type of terminal. Also, you can set the EDT keypad or WPS keypad on either type of terminal.
- **SET NOCLIPBOARD**
  Copy, cut, and paste operations use the Insert Here buffer in EVE. On DECwindows, you can enable the clipboard, which lets you transfer text between EVE and other DECwindows applications. WPS keypad keys do not use the clipboard, regardless of the setting.
- **SET NODEFAULT COMMAND FILE**
  EVE uses one of the following as the default command file for saving attributes:
  - Command file specified with the /COMMAND= qualifier when you invoked EVE
  - Command file named TPU$COMMAND.TPU in the current directory
  - A command file defined by the logical name TPU$COMMAND

- **SET NODEFAULT SECTION FILE**
  If section file prompting is enabled (the default), EVE prompts whether to save attributes in a section file. If section file prompting is disabled, EVE prompts whether to save attributes in a command file.
- **SET NOGOLD KEY**
  EVE does not have a default GOLD key. Setting the EDT or WPS keypad makes PF1 the GOLD key, overriding any current definition of PF1, unless you set a different key as GOLD.
- **SET NOPENDING DELETE**
  Using DELETE or typing new text does *not* erase a selection.
- **SET SECTION FILE PROMPTING**
  When you save attributes and other customizations, EVE prompts for a section file.
- **SET SCROLL MARGINS 0 0**
  Scrolling begins automatically when you move past the top or bottom of the window.
- **SET TABS INSERT**
  Using TAB inserts a tab character. You can set the tab mode to insert spaces instead of a tab character or to move the cursor without inserting anything.
- **SET TABS INVISIBLE**
  Tab characters appear during editing as blank spaces, as opposed to visible tabs, which appear as a small H_T (horizontal tab).
- **SET WIDTH 80**
  The width of the EVE screen layout is the same as your terminal setting—typically 80 columns.
- **SET WILDCARDS VMS**
  The WILDCARD FIND command uses wildcards such as the asterisk (*) to match any amount of text on a line, the percent sign (%) to match a single character on a line, and so on.

A.4.3 EVE Default Buffer-Specific Settings

This list shows the EVE default settings for buffer-specific settings:

- **FORWARD**
  Commands like FIND and MOVE BY LINE move the cursor to the right and down. You can change the direction to reverse (left and up).
- **INSERT MODE**
  Characters you type are inserted at the current position, pushing existing text to the right and down. You can change the mode to overstrike.

- **SET BUFFER MODIFIABLE**
  Buffers you create can be modified (edited). You can set the buffer to unmodifiable.
- **SET BUFFER WRITE**
  On exiting, EVE writes out (saves) your buffers if you have made any changes. You can set the buffer to read-only.
- **SET JOURNALING ALL**
  Buffer-change journaling is enabled for all your text buffers.
- **SET LEFT MARGIN 1**
  This is the leftmost column. When you press the Return key or use FILL commands or when EVE wraps text, new lines start at the left margin of the buffer.
- **SET PARAGRAPH INDENT 0**
  Paragraphs you create or ones you reformat with FILL commands start at the current left margin of the buffer—with no indent.
- **SET RIGHT MARGIN 79**
  The default right margin is one column less than the width set for your terminal. If the width is 80 columns, the default right margin is 79. When you use FILL commands or when you type at the end of a line, EVE wraps text at the right margin of the buffer.
- **SET TABS EVERY 8**
  Tab stops are set at columns 9, 17, 25, 33, 41, and so on. You can set tab stops at different intervals.
- **SET WRAP**
  As you type text at the end of a line, EVE wraps text at the right margin of the buffer, without your having to press the Return key or use FILL commands.

A.4.4 Default Direction

When editing EVE command lines—such as when you recall a command—the default direction is reversed and the cursor is bound. The default mode on a character-cell terminal matches your terminal setting.

To find out the default settings, use the SHOW DEFAULTS BUFFER command. To find out the settings of the buffer you are editing, use the SHOW command.

A.4.5 Saving Attributes

You can save some EVE settings or attributes in a section file or as EVE-generated code in a DECTPU command file. You can set other attributes, such as margins and tab stops, in an initialization file.

A.4.6 Initialization Files

When you use an initialization file to invoke EVE, commands in the initialization file for margins, tab stops, and other buffer-specific settings apply to the Main (or first) buffer and to an EVE system buffer named $DEFAULTS$. The $DEFAULTS$ buffer is a template buffer; when you create a buffer—for example, by using the GET FILE command—EVE uses the settings of the $DEFAULTS$ buffer so that each new buffer has the same settings. Thus, if your initialization file contains the command SET RIGHT MARGIN 70, each buffer you create will have that right margin.

A.4.7 Example

This is a sample EVE initialization file that contains commands to set editing preferences and to define keys:

```
!   MYINIT.EVE initialization file
!
SET LEFT MARGIN 5
SET PARAGRAPH INDENT 4
SET RIGHT MARGIN 70
SET TABS EVERY 10
SET SCROLL MARGINS 9% 9%
SET FIND WHITESPACE
!   Key definitions
SET KEYPAD EDT
DEFINE KEY= F20      SHOW BUFFERS
DEFINE KEY= Ctrl/P   PAGINATE
DEFINE KEY= GOLD-G   GET FILE
DEFINE KEY= KP7      WPS GOLD R
```

A.4.8 Saving Attributes

Attributes are global settings, some of which can be saved in a section file or DECTPU command file for future editing sessions. Table A–1 shows the settings that you can save:

Table A–1 EVE Commands for Setting Attributes

Command	Default Setting
SET BOX [NO]PAD	SET BOX PAD
SET BOX [NO]SELECT	SET BOX NOSELECT
SET [NO]CLIPBOARD	SET NOCLIPBOARD
SET CURSOR { BOUND | FREE }	SET CURSOR FREE
SET [NO]DEFAULT COMMAND FILE	SET NODEFAULT COMMAND FILE
SET [NO]DEFAULT SECTION FILE	SET NODEFAULT SECTION FILE

(continued on next page)

Table A–1 (Cont.) EVE Commands for Setting Attributes

Command	Default Setting
SET FIND CASE [NO]EXACT	SET FIND CASE NOEXACT
SET [NO]EXIT ATTRIBUTE CHECK	SET EXIT ATTRIBUTE CHECK
SET [NO]SECTION FILE PROMPTING	SET SECTION FILE PROMPTING
SET [NO]PENDING DELETE	SET NOPENDING DELETE
SET TABS { INSERT / MOVEMENT / SPACES }	SET TABS INSERT
SET TABS [IN]VISIBLE	SET TABS INVISIBLE

If you have an EVE initialization file that contains commands for these settings, you can delete those command lines after you save the settings in your section file or command file.

Other global settings (such as scroll margins or the types of wildcards) and any buffer settings (such as margins or tab stops) are not saved. Typically, you use an initialization file for those settings.

A.4.9 EVE Commands for Saving Attributes

The following list summarizes the new and changed commands for saving attributes:

- **SAVE ATTRIBUTES**
  Saves attributes in a section file or command file, depending on your responses to EVE prompts or settings done with other EVE commands. If you save in a section file, the effect is the same as SAVE EXTENDED EVE. If you save in a command file, EVE generates a specially marked block of DECTPU statements for attribute settings and menu definitions, and either creates a command file or updates an existing command file with this block of statements.
- **SAVE SYSTEM ATTRIBUTES**
  Saves EVE default attributes in a section file or command file. This is useful if you want to restore your section file or command file to the standard EVE settings and menu definitions (see Section A.4.22).
- **SAVE EXTENDED EVE**
  Creates a section file, saving attributes, key definitions, menu definitions, compiled procedures, and other extensions, such as global variables set with a DECTPU statement. If you do not

specify a section file on the command line, EVE prompts you for one or uses your default section file (if you set a default).

- **SET BOX NOPAD**
  Disables padding and overstriking for box editing, unless the mode of the buffer is overstrike.
- **SET BOX NOSELECT**
  Disables box-style selection, cutting, and pasting. Default setting.
- **SET BOX PAD**
  Enables padding and overstriking for box editing, regardless of the mode of the buffer. Default setting.
- **SET BOX SELECT**
  Enables box selection, cutting, and pasting.
- **SET DEFAULT COMMAND FILE**
  Determines the command file for saving attributes. Does not determine the command file to be executed at startup, if any.
- **SET DEFAULT SECTION FILE**
  Determines the section file for saving attributes. Does not determine the section file to be executed at startup.
- **SET EXIT ATTRIBUTE CHECK**
  If you have changed attributes and then exit or quit, EVE asks if you want to save your changes. Default setting.
- **SET NODEFAULT COMMAND FILE**
  When you save attributes, the default command file is TPU$COMMAND.TPU in your current directory or the command file that was executed at startup (see Section A.4.17). Default setting.
- **SET NODEFAULT SECTION FILE**
  When you save attributes, EVE asks for the name of the section file you want to create (unless you disabled section file prompting). Default setting.
- **SET NOEXIT ATTRIBUTE CHECK**
  Disables attribute checking, typically to speed up or simplify exiting or quitting. Does not apply to the editing session in which you issue the command. Applies only to the editing sessions in which you use the section file or command file in which you saved the setting.
- **SET NOSECTION FILE PROMPTING**
  Disables prompting for a section file when you save attributes, typically to speed up or simplify saving attributes in a default section file or in a command file.
- **SET SECTION FILE PROMPTING**
  When you save attributes, EVE prompts you for the name of a section file. Default setting.

A.4.10 Saving Attributes While Editing

You can save attributes during your editing session by using the SAVE ATTRIBUTES or SAVE EXTENDED EVE command or as part of exiting or quitting. By default, if you have changed attributes and not saved them, then on exiting EVE prompts you as follows:

```
Command: SET CURSOR BOUND
Command: SET FIND CASE EXACT
Command: SET TABS VISIBLE
            .
            .
            .
Command: EXIT
Attributes were changed.  Save them? [YES]
```

If you want to save the changes, press the Return key. EVE then executes a SAVE ATTRIBUTES command before continuing the exit. If you do not want to save the changes, type No and press the Return key. EVE then continues exiting.

A.4.11 Disabling Prompting

To disable this prompting—typically, to make exiting faster or simpler—use the SET NOEXIT ATTRIBUTE CHECK command. However, the command does not apply to the current editing session because exit checking is itself a global setting that you can save in a section file or command file. After you save it, the setting applies to future editing sessions in which you use the relevant section file or command file.

A.4.12 Saving Attributes in a Section File

Typically, you save attributes in a section file. A section file is in binary form and saves attributes, key definitions (including learn sequences), menu definitions, compiled procedures, and other extensions to the editor—including any saved in the section file you are using. In effect, the section file is your customized version of EVE. Because the section file is binary, it is executed quickly at startup.

A.4.13 Creating Section Files

To create a section file, you can use the SAVE EXTENDED EVE command (as in previous versions of EVE) or the SAVE ATTRIBUTES command. When using SAVE EXTENDED EVE, you can specify the section file on the command line or let EVE prompt you for the section file name. When using SAVE ATTRIBUTES, you specify the section file as a response to a prompt.

To speed up saving in a section file, you can set a default section file, which you can then save in without having to specify the file each time you save attributes. You can also disable section file prompting.

A.4.14 Example: Creating Section Files

The command shown in this example saves attributes and other customizations in a section file called MYSEC.TPU$SECTION in the current directory:

```
Command: SAVE ATTRIBUTES
Save attributes in a section file [YES]? Return
File to save in: mysec
DISK$1:[USER]MYSEC.TPU$SECTION;1 created
```

A.4.15 EVE Settings for Saving Attributes

The following table shows the interaction of the settings for default section file and section file prompting:

Commands Settings	Effect with SAVE ATTRIBUTES
SET DEFAULT SECTION FILE SET SECTION FILE PROMPTING	When you save attributes, EVE asks you whether to save in a section file. If you respond Yes (the default response), EVE saves in your default section file. If you respond No, EVE asks whether to save in a command file.
SET DEFAULT SECTION FILE SET NOSECTION FILE PROMPTING	When you save attributes, EVE saves in your default section file without prompting.
SET NODEFAULT SECTION FILE SET SECTION FILE PROMPTING	When you save attributes, EVE asks whether to save in a section file. If you respond Yes, EVE asks for the name of a section file. If you respond No, EVE asks whether to save in a command file. Default settings.
SET NODEFAULT SECTION FILE SET NOSECTION FILE PROMPTING	When you save attributes, EVE asks whether to save in a command file (see Section A.4.17).

A.4.16 Specifying Section Files

Typically, when you use SET DEFAULT SECTION FILE, you specify the section file you are going to use at startup for future editing sessions. The command does not determine the section file to be executed when you invoke the editor, but only the section file in which you save attributes and other customizations.

Section files may be quite large, depending on the number of key definitions, menu definitions, and procedures you save. If you have limited disk space, you should save in a command file, which requires less disk space. For more information about creating and using section files, see the EVE online help topic called Section Files.

A.4.17 Saving Attributes in Command Files

A command file contains DECTPU procedures and statements that are compiled and executed at startup. In effect, this is a series of programs for extending EVE. (You can also use a command file for batch editing.) A command file may be slower at startup than a section file (depending on the number of procedures to be compiled and statements to be executed), but it takes up less disk space than a section file. In addition, a command file can be edited and printed. Also, if you edit your command file, you can recompile procedures during your editing session by using EXTEND commands. The default file type for command files is .TPU.

When you use the SAVE ATTRIBUTES command or when you save attributes on exiting or quitting, you can have EVE create or update a command file. EVE then generates a specially marked block of DECTPU statements for your settings and menu definitions. Thus, if you created a command file with procedures and key definitions of your own, you can have EVE append the block of attribute settings to this command file.

A.4.18 Example: EVE Generated Code

Example A–1 is an example of EVE-generated code for saving attributes in a command file.

Example A–1 EVE-Generated Code for Saving Attributes in a Command File

```
! EVE-generated code begin
! EVE attributes begin
eve$set_find_case_sensitivity (FALSE);
eve_set_box_noselect;
eve_set_box_pad;
eve_set_cursor_bound;
eve_set_nodefault_command_file;
eve_set_nodefault_section_file;
eve_set_exit_attribute_check;
eve_set_pending_delete;
eve_set_nosection_file_prompting;
eve_set_tabs ('INSERT');
eve_set_tabs ('VISIBLE');
! EVE attributes end
! EVE-generated code end
```

A.4.19 Example: Saving Attributes in Command Files

To save attributes in a command file, use the SAVE ATTRIBUTES command, as follows:

```
Command: SAVE ATTRIBUTES
Save attributes in a section file [YES]? no
Save attributes in a command file [YES]? Return
Enter file name [TPU$COMMAND.TPU] MYCOM
14 lines written to file DISK$1:[USER]MYCOM.TPU;1
```

A.4.20 Default Command Files

The prompt for the command file name shows, in brackets, the default command file that EVE uses if you press the Return key at the prompt without typing a file name. This default is one of the following:

- The command file specified with the /COMMAND qualifier when you invoked EVE
- A file called TPU$COMMAND.TPU in your current directory
- The command file defined by the logical name TPU$COMMAND

A.4.21 Setting a Default Command File

You can set your preferred default command file—that is, the command file you want EVE to create or update without having to specify the file each time you save attributes. For example, the following command sets your default command file as MYCOM in your current directory:

```
Command: SET DEFAULT COMMAND FILE MYCOM
```

If you want to save in a command file rather than in a section file, you should also use the SET NOSECTION FILE PROMPTING command. Then, when you save attributes, EVE asks whether to save in a command file without first asking whether to save in a section file.

Typically, when you use SET DEFAULT COMMAND FILE, you specify the command file you are going to use at startup for future editing sessions. The command does not determine the command file to be executed when you invoke EVE, but only the command file in which you save attributes and menu definitions.

For more information about creating and using command files, see the EVE online help topic called Command Files.

A.4.22 Saving EVE Default Attributes

The SAVE SYSTEM ATTRIBUTES saves EVE default settings and menu entries in a section file or command file. Thus, if you set several attributes and defined or undefined menu entries, you can use SAVE SYSTEM ATTRIBUTES to restore the standard EVE settings and menus to your section file or command file.

SAVE SYSTEM ATTRIBUTES does not change the settings currently in effect—for example, it does not enable free cursor motion or invisible tabs—but saves only the EVE defaults in a section file or command file.

A.5 Using DECTPU Within EVE

A.5.1 Overview

You can use DECTPU within EVE to create DECTPU command files and to use the DECTPU debugging package.

File creation switches and qualifiers determine whether DECTPU creates a buffer when it does not find the input file. The processing results of using this qualifier depend on the DECTPU application you are using.

In EVE, files are created by default. If the input file does not exist, EVE uses the input file name and file type to create the buffer name. If you do not specify an input file, EVE creates a buffer named Main.

A.5.2 Creating DECTPU Command Files

To create DECTPU command files, use the /CREATE or /NOCREATE qualifiers, as follows:

```
$ EDIT/TPU /CREATE (default)
$ EDIT/TPU /NOCREATE
```

Use the /NOCREATE qualifier to avoid invoking the editor in case you mistype the input file specification or to edit only an existing file.

If EVE does not find an input file you have specified, it terminates the editing session and returns you to the system level, as in the following example:

```
$ EDIT/TPU NEW.DAT /NOCREATE
Input file does not exist: NEW.DAT;
```

A.5.3 Using a DECTPU Debugging Package

Debug switches and qualifiers determine whether DECTPU runs a debug file. A debug file is useful for testing procedures for an application that you are creating.

To edit the code in the file you are debugging, follow these rules:

1. Use the GO command. You cannot use wildcards to specify the debug file.
2. Specify only one debug file at a time. DECTPU compiles and executes the debug file before executing TPU$INIT_ PROCEDURE.

The debugger that is supplied with DECTPU is in SYS$SHARE:TPU$DEBUG.TPU. This file provides commands to manipulate variables and to control program execution.

A.5.4 Specifying a Debug File

There are two ways to specify a debug file of your own:

- Define the logical name TPU$DEBUG to specify your debugger file and then use the /DEBUG qualifier when you invoke DECTPU. You can enter the definition in your LOGIN.COM file.
- Use the /DEBUG= qualifier and specify a debugger file.
  This overrides any definition of the TPU$DEBUG logical name. The default file type is .TPU. For example, the following command uses a debugger file named MYDEBUG.TPU to edit a file named MYPROCS.TPU:

```
$ EDIT/TPU MYPROCS.TPU /DEBUG=MYDEBUG
```

DECTPU assumes the debugger file is in SYS$SHARE. If your debugger file is stored elsewhere, use a complete file specification, including the device (disk) and directory.

For more information about the DECTPU debugging package, read the comments in the source file or see the *DEC Text Processing Utility Reference Manual*.

A.6 Using DECTPU Procedures to Extend EVE

A.6.1 Overview

The EVE editor is built on the DEC Text Processing Utility (DECTPU). With the EVE command TPU, you can enter any DECTPU statement or series of statements that can be expressed on one command line.

A.6.2 Entering DECTPU Commands

To enter a DECTPU command, enter the command TPU followed by the DECTPU statement you want to execute. For example, to execute the DECTPU statement APPEND_LINE, which places the current line at the end of the previous line, enter the following command string:

```
Command: TPU APPEND_LINE
```

For more information about the TPU command, type HELP TPU. See the *DEC Text Processing Utility Reference Manual* for a complete list of DECTPU statements and procedures.

A.6.3 Writing DECTPU Procedures

Because EVE is an editor written in the DECTPU programming language, you can extend EVE functions by writing procedures in DECTPU. This section assumes that you are familiar with the DECTPU programming language described in the *DEC Text Processing Utility Reference Manual*.

Before you begin writing DECTPU procedures to modify EVE, Digital recommends that you study the EVE source code, which is stored in SYS$EXAMPLE:EVE$*.TPU. (The wildcard character (*) in the file specification indicates that EVE source code is stored in many files.) These files are put together by using EVE$BUILD. Variables and statements in your procedures should be consistent with EVE variables and statements so that you can avoid making changes that adversely affect EVE operations.

A.6.4 Rules for Writing EVE Command Procedures

You can write procedures in the DECTPU programming language that are, in effect, new EVE commands. When you write new EVE command procedures, follow these rules:

- Prefix the procedure name with the label *EVE_* so that EVE recognizes the procedure as an EVE command. After you compile the procedure, execute it by pressing the Do key and typing the procedure name without the EVE_ prefix or the underscores. For example, enter SET LEFT MARGIN for the

EVE_SET_LEFT_MARGIN procedure. (You can also define a key to execute the new command.)

- The words *PROCEDURE* and *ENDPROCEDURE* must start in column 1.
- Initialize all global variables in a single procedure named TPU$LOCAL_INIT or in a module initialization procedure if you build with EVE$BUILD.
- Define a global variable to associate the parameter with the data-type integer if the EVE command you are writing takes any integer parameters (such as SET LEFT MARGIN 2, where 2 is the parameter). When you are using the EVE commands that you have defined, EVE passes the null string (" ") if you do not provide a value for a string parameter. If you do not provide a value for an integer parameter, EVE passes the value EVE$K_NO_ARG.
- Digital recommends placing global variables in a procedure called TPU$LOCAL_INIT, which is called each time EVE starts. Place all procedures that use parameters in the same file that contains the TPU$LOCAL_INIT procedure.
- In general, each of the global variable names that define command parameters must consist of the following three parts:
  - The first part of the variable name must be *EVE$*.
  - The second part defines the variable's sequence within a procedure. For the first variable, it is *ARG1*; for the second, *ARG2*; and so on.
  - The third part of the variable name is the procedure name without the *EVE_* prefix.

A.6.5 Example: Defining a Global Variable

This is an example of a global variable definition for command parameters:

- The following definition of the global variable EVE$ARG1_ ADD tells EVE to expect an integer as the first parameter for EVE_ADD:

```
EVE$ARG1_ADD := "INTEGER";
```

- The following definition of the global variable EVE$ARG1_ HELLO tells EVE to expect a string as the first parameter to EVE_HELLO:

```
EVE$ARG1_HELLO := "STRING";
```

The integer variables are required.

A.6.6 Compiling DECTPU Procedures

The EXTEND EVE command enables you to compile a DECTPU procedure without leaving EVE. To compile the procedure that the cursor is in, use the EXTEND THIS command. To compile one procedure, enter the command EXTEND EVE and the name of the procedure you want to compile. To compile all procedures in a file, enter the command EXTEND EVE * (which is the same as the EVE command EXTEND ALL). If you miss a message from the compiler, use the command BUFFER MESSAGES to read the messages stored in the Messages buffer.

A.6.7 Example: Creating DECTPU Procedures

The following example illustrates how to create and compile DECTPU procedures to define an ADD command, a HELLO command, and the parameters for both commands. Invoke EVE to edit the file MYPROCEDURES.TPU and insert the following text into the file:

```
! Procedure to add two integers and display the result in the
! message window

PROCEDURE EVE_ADD (A1, A2)

LOCAL   TEMP,
        N1,
        N2;

IF NOT EVE$PROMPT_NUMBER (A1, N1, "First number to add: ",
                          "No number specified.")
THEN
    RETURN (FALSE);
ENDIF;

IF NOT EVE$PROMPT_NUMBER (A2, N2, "Second number to add: ",
                          "No number specified.")
THEN
    RETURN (FALSE);
ENDIF;

TEMP := N1 + N2;
MESSAGE (STR (N1) + " + " + STR (N2) + " = " + STR (TEMP));
RETURN (TRUE);

ENDPROCEDURE;


PROCEDURE EVE_HELLO (MY_NAME)
LOCAL   THE_NAME;

IF EVE$PROMPT_STRING (MY_NAME, THE_NAME, "Name: ", "We haven't met")
THEN
    MESSAGE ("Hello " + THE_NAME);
    RETURN (TRUE);
ELSE
    RETURN (FALSE);
ENDIF;

ENDPROCEDURE;

EVE$ARG1_ADD := "INTEGER";
EVE$ARG2_ADD := "INTEGER";
```

Use the syntax shown in the file MYPROCEDURES.TPU to put the definitions for the two parameter variables in your TPU$LOCAL_INIT procedure.

To compile the procedures you have entered into MYPROCEDURES.TPU, press the Do key, type EXTEND EVE *, and press the Return key. If you are going to use the newly compiled commands in the current editing session, you must execute TPU$LOCAL_INIT by entering the command TPU TPU$LOCAL_INIT.

A.7 Creating Section Files

A.7.1 Overview

A section file contains key definitions, learn sequences, and compiled DECTPU statements and procedures in binary form. Because section files are in binary form, they set up the editing environment very quickly, but you cannot display or edit a binary file. Use a section file to implement editing features that are not likely to change from one editing session to another. The default file type for section files is .TPU$SECTION.

A.7.2 Startup Section Files

EVE requires a section file for startup. By default, EVE uses the section file EVE$SECTION.TPU$SECTION located in directory SYS$SHARE. This default section file defines the editing keys, shown in Figure 8–1 and Figure 8–2, as well as the standard EVE commands.

A.7.3 Modifying Section Files

Rather than use the default section file, you can create a modified section file that contains the standard EVE functions as well as your own key definitions, learn sequences, and editing functions. You can create a section file in two ways:

- If you have a small number of key definitions and learn sequences, define editing keys and assign learn sequences interactively. To save the definitions in a section file, enter the SAVE EXTENDED EVE command.

  The default file type for section files is .TPU$SECTION. Each time you enter the SAVE EXTENDED EVE command, you can specify the same file to ensure that all your customizations are saved cumulatively in the same file.

- If you have a large number of modifications, create a command file with key definitions and DECTPU procedures. (Note, however, that a command file cannot have learn sequences.) End the file with the SAVE and QUIT statements, assigning a section file name with the file type .TPU$SECTION, as shown in Section A.8.8. Then invoke EVE with the /COMMAND qualifier and the command file name. EVE executes statements in the command file, saves the compiled procedures and key definitions in the section file named in the SAVE statement, and then executes the QUIT statement, returning control to the DCL. You can now use the new section file, as named in the SAVE statement.

A.7.4 Using Section Files

To use a section file, specify the section file name with the /SECTION qualifier on the EVE command line. For example, the following command invokes EVE with a section file named MY_SECTION.TPU$SECTION, located in directory ALEXIS on a disk called WORK1:

```
$ EDIT/TPU/SECTION=WORK1:[ALEXIS]MY_SECTION
```

A.7.5 Default Section Files

By default, DECTPU uses the section file whose logical name is TPU$SECTION. If you define this logical name in your LOGIN.COM file, DECTPU automatically uses your section file when you invoke EVE. For example:

```
$ DEFINE TPU$SECTION WORK1:[ALEXIS]MY_SECTION.TPU$SECTION
```

Use the EVE command SHOW SUMMARY to display the name of the current section file.

A.7.6 Section Files Execution

EVE executes a section file before a command file or an initialization file. Therefore, definitions in the command file and initialization file override section file definitions. When you want to set the characteristics of the editing environment, use either a command file or an initialization file. EVE executes these commands upon startup, so the appearance of the buffer and the editing mode is adjusted according to your definition.

A.8 Creating Command Files

A.8.1 Overview

A command file is a source program that contains DECTPU procedures and executable statements. A DECTPU procedure is a set of related DECTPU statements that are executed when the procedure name is invoked. The statements and procedures define what happens when you press a key or enter a command. The default file type for command files is .TPU.

When you use an EVE command, you are actually invoking a compiled DECTPU procedure. For example, the EVE command SET KEYPAD EDT invokes the EVE_SET_KEYPAD_EDT procedure in the standard EVE section file.

EVE executes a command file after a section file. For this reason, any key definitions or procedures defined in a command file override those in a section file.

A.8.2 Command File Usage

There are two different ways to use a command file. A command file can create an editing environment that is independent of the section file, or a command file can be used to produce a new section file.

A.8.3 Setting Editing Defaults

Whenever you want to set editing defaults, use a command file because EVE executes the statements in the command file (or initialization file) at startup and applies the new defaults. See Section A.9 for a list of commands that set the editing environment. For example, you can use one command file to set up margins and tabs and a header for a memo and another command file to set tabs suitable for writing a financial report.

To create a command file, invoke EVE and specify a file name with the file type .TPU, such as MY_COMMAND.TPU. Once in the editor, enter DECTPU statements and procedures.

A.8.4 Converting Command Files

If you intend to use a command file to create a section file, conclude the file with the SAVE and QUIT statements and include a file specification for the section file, as shown in Section A.8.8. To convert the command file to a section file, invoke EVE with the /COMMAND qualifier. For example:

```
$ EDIT/TPU/COMMAND=MY_COMMANDS
```

EVE executes the commands in the file and saves the compiled procedures and key definitions in the TPU$SECTION file that you name in the SAVE statement. The QUIT statement terminates the editor, returning control to the DCL prompt. At this point, you have a new section file. Therefore, on invoking EVE, the editor automatically reads the section file that you have defined in your LOGIN.COM or in your SYSLOGIN directory.

One advantage of creating a section file from a command file is that a command file can be easily edited. This is especially important when you want to add DECTPU procedures or add a large number of key definitions.

A.8.5 Adding Functions to the EVE Editor

To use a command file to set the characteristics of the editing environment and add functions to the standard EVE editor, again invoke EVE with the /COMMAND qualifier. For example:

```
$ EDIT/TPU/COMMAND=DATA_SETUP
```

EVE again executes the statements in the command file but because there is no SAVE statement, the compiled procedures and key definitions are not saved.

A.8.6 Default Files Specification

If you do not include a command qualifier, EVE searches for the file specified by the logical name TPU$COMMAND. You can define this logical name in your LOGIN.COM file. If this file is not found, DECTPU then searches for a file named TPU$COMMAND.TPU in the current directory.

A.8.7 Rules for Writing Command Files

Keep in mind the following rules and suggestions when writing command files:

- Use comments to document a command file. An exclamation point (!) starts a comment. When DECTPU encounters the exclamation point, it ignores everything else on the line.
- Start each procedure with the word ‘‘procedure’’.
- End each DECTPU statement in the procedure with a semicolon (;).
- Make each EVE command correspond to the name of a DECTPU procedure in the section file of standard EVE (SYS$SHARE:EVE$SECTION.TPU$SECTION).

  You can use the names of these DECTPU procedures in procedures you write, or you can execute them independently as DECTPU statements. If your command file contains both user-written procedures and executable DECTPU statements, put all the procedures before any of the executable statements.
- End each procedure with the word ‘‘endprocedure’’.
- Use EVE_ as the first four characters of the procedure name when you write a procedure that implements a command in EVE, By following this convention, the procedure name, without the characters EVE_, becomes an EVE command.

  Because a command file executes after a section file, a procedure overwrites a command by the same name in the section file. For example, if you name a procedure EVE_ ERASE_CHARACTER, the ERASE CHARACTER command executes your procedure, not the standard EVE command.
- Place parameters in parentheses if you use a DECTPU procedure name that requires parameters. The EVE format is as follows:

  ```
  SET LEFT MARGIN 10
  ```

  The DECTPU format is as follows:

  ```
  eve_set_left_margin(10);
  ```
- If a parameter is a string rather than an integer, enclose the parameter in quotation marks (" ") To pass a null parameter, use an empty pair of quotation marks (" ").
- Compile procedures before invoking them. Then the procedure can be invoked as either an EVE command or as a DECTPU executable statement.
- Add a procedure called TPU$LOCAL_INIT to a command file that you use as a section file. This procedure should contain all executable statements (including those calling other procedures) that you want to be defined and executed when EVE is invoked. Because any executable statement in this procedure is executed when EVE is invoked, the statements in this procedure become default settings for the EVE editor.

- Add the statement QUIT as the last statement in a command file if the file is to be compiled as a section file. QUIT ends EVE processing and returns control to the DCL prompt.

A.8.8 Example: EVE Command File

The following example shows a command file that modifies EVE to be more like the EDT editor. This file creates a personal section file named MY_SECTION.TPU$SECTION.

```
!*********************************************************************
!Command file making EVE more like EDT
!and implementing personal customizations
!*********************************************************************

!Procedure to delete a line and close the gap left by the deletion  (1)

PROCEDURE EVE_ZAPLINE

EVE_END_OF_LINE;

EVE_ERASE_START_OF_LINE;      (2)

EVE_DELETE;

ENDPROCEDURE

!Procedure to move the cursor to the beginning of the next paragraph:

PROCEDURE EVE_NEXT_PARAGRAPH  (3)

LOCAL   PAT1,
        THE_RANGE;

PAT1 := line_begin + line_begin + ARB (1);
THE_RANGE := SEARCH_QUIETLY (PAT1, forward, exact);

IF THE_RANGE <> 0
THEN
    POSITION (END_OF (THE_RANGE));
    RETURN (TRUE);            (4)
ELSE
    RETURN (FALSE);
ENDIF;

ENDPROCEDURE

!Procedure to make EVE behave more like EDT

PROCEDURE EVE_MIMIC_EDT

EVE_SET_KEYPAD_EDT;
EVE_SET_CURSOR_BOUND;
EVE_SET_LEFT_MARGIN(10);      (5)

ENDPROCEDURE

!Procedure to transpose two characters

PROCEDURE EVE_TRANSPOSE

LOCAL   WHACK;

WHACK := ERASE_CHARACTER (1);
MOVE_HORIZONTAL (1);
COPY_TEXT (WHACK);

RETURN (TRUE);
ENDPROCEDURE
```

```
!Procedure to make both the screen width and the right margin narrow

PROCEDURE EVE_NARROW_SCREEN

EVE_SET_WIDTH (80);
EVE_SET_RIGHT_MARGIN (79);

ENDPROCEDURE;

!Procedure to make both the screen width and the right margin wide

PROCEDURE EVE_WIDE_SCREEN

EVE_SET_WIDTH (132);
EVE_SET_RIGHT_MARGIN (131);

ENDPROCEDURE;

!Procedure to toggle screen width and right margin from
!the current setting to the other setting, for example to
!change to wide if narrow, change to narrow if wide

PROCEDURE EVE_CHANGE_WIDTH    ❻

IF GET_INFO (SCREEN, "width") <> 80
THEN
    EVE_NARROW_SCREEN;
ELSE
    EVE_WIDE_SCREEN;
ENDIF;

ENDPROCEDURE;

PROCEDURE TPU$LOCAL_INIT      ❼

EVE_MIMIC_EDT;                ❽

EVE$DEFINE_KEY ("EVE_NEXT_PARAGRAPH", CTRL_P_KEY, "Next Para",
                EVE$X_USER_KEYS); ❾

EVE$DEFINE_KEY("EVE_ZAPLINE", KEY_NAME ("0", SHIFT_KEY), "Zap Line",
               EVE$X_USER_KEYS);  ❿

EVE$DEFINE_KEY ("EVE_TWO_WINDOWS", F17, "Two Windows", EVE$X_USER_KEYS);
                                                                    ⓫

EVE$DEFINE_KEY ("EVE_OTHER_WINDOW", CTRL_G_KEY, "Other Window",
                EVE$X_USER_KEYS); ⓬

EVE$DEFINE_KEY ("EVE_GET_FILE('')", KEY_NAME (KP6, SHIFT_KEY), "Get File",
                EVE$X_USER_KEYS); ⓭

EVE$DEFINE_KEY ("EVE_TRANSPOSE", KEY_NAME (F20, SHIFT_KEY), "Transpose",
                EVE$X_USER_KEYS); ⓮
ENDPROCEDURE

TPU$LOCAL_INIT;               ⓯

SAVE ("WORK:[LINCOLN]MY_SECTION.TPU$SECTION");   ⓰

QUIT;                         ⓱
```

As you examine the example, note the following:

❶ This line is preceded by an exclamation point (!), thus DECTPU ignores everything else on the line.

❷ Each EVE command corresponds to the name of a DECTPU procedure in the section file of standard EVE (SYS$SHARE:EVE$SECTION.TPU$SECTION).

You can use the names of these DECTPU procedures in procedures you write, or you can execute them independently as DECTPU statements. If your command file contains both user-written procedures and executable DECTPU statements, put all the procedures before any of the executable statements.

❸ This command calls the EVE procedure EVE_NEXT_ PARAGRAPH. Once the section file is compiled, you can use the new EVE command NEXT PARAGRAPH.

❹ For an EVE command to be usable with a repeat count (either by itself or as part of a learn sequence), it must return TRUE when it succeeds.

❺ The EVE command SET LEFT MARGIN requires a parameter specifying where to set the left margin. In this example, it is set to 10.

❻ The procedure named EVE_CHANGE_WIDTH can be invoked as the EVE command CHANGE WIDTH. The procedure name can also be invoked as the DECTPU executable statement EVE_CHANGE_WIDTH.

❼ The procedure called TPU$LOCAL_INIT is used as a section file.

❽ This DECTPU statement invokes the procedure EVE_MIMIC_ EDT, which contains DECTPU statements that change the EVE settings. If you compile the sample command file, save it in a personal section file, and use that section file to invoke EVE. By doing this, the keypad setting, cursor style, and left margin are automatically set by the procedure EVE_MIMIC_ EDT. As a result, EVE behaves like EDT at startup.

❾ This DECTPU statement uses the predefined EVE routine EVE$DEFINE_KEY to define the user-written procedure EVE_NEXT_PARAGRAPH for the key sequence Ctrl/P. The EVE$DEFINE_KEY routine ensures that the program bound to the key has an error handler. The DECTPU built-in DEFINE_KEY does not perform this step.

There are four parameters to EVE$DEFINE_KEY. (This routine uses the same parameters as the DECTPU built-in DEFINE_KEY.) The first specifies the EVE procedure or command to be bound to a key. The second specifies the key to which the command should be bound. The third specifies the label that EVE should use for the key in the Help keypad diagram. The fourth is an EVE variable specifying the keymap list in which the key definition should be saved. Use the variable name EVE$X_USER_KEYS for the fourth parameter unless you are an advanced user implementing a special application.

⓾ This DECTPU statement defines the procedure EVE_ZAPLINE for the key sequence GOLD O. This line demonstrates the DECTPU format to use when defining a sequence that consists of the GOLD key plus a letter key. Using ''shift_key'' makes the definition case insensitive, so that both *o* and *O* are defined.

⓫ This statement defines the EVE command TWO WINDOWS for the F17 key.

⓬ This statement defines the EVE command OTHER WINDOW for the key sequence Ctrl/G.

⓭ This statement defines the EVE command GET FILE for the key sequence GOLD KP6. This keypad binding supersedes the previous definition of the GOLD KP6 key. The EVE_MIMIC_EDT procedure sets the keypad to EDT and the EDT keypad then binds the GOLD KP6 key sequence to the INSERT HERE command. However, this DECTPU statement changes the key binding to the GET FILE command. The pair of single quotation marks passes a null argument to the procedure.

⓮ This statement defines the procedure EVE_TRANSPOSE for the key sequence GOLD F20.

⓯ This statement calls the procedure TPU$LOCAL_INIT, which is then executed, creating new default settings for EVE.

⓰ The SAVE statement is included to create a new section file. The device, directory, and file name of the section file are specified in parentheses and quotation marks.

⓱ The QUIT command ends EVE processing and returns control to the DCL prompt.

A.9 Creating Initialization Files

A.9.1 Overview

Rather than defining keys or setting the characteristics of an editing session interactively, you can put EVE commands and key definitions in an initialization file. You can execute an initialization file when invoking EVE or during an editing session by using the execute procedure (@) command. For example,

```
Command: @SETUP_INIT
```

A.9.2 Rules for Creating Initialization Files

The following rules apply when creating initialization files:

- Begin each command on a separate line.
- Precede each line of comments with an exclamation point (!) and place separately from command lines.
- Give the initialization file the file type .EVE.

A.9.3 Example

This is an example of an initialization file:

```
SET TABS EVERY 5
SET LEFT MARGIN 15
SET RIGHT MARGIN 75
OVERSTRIKE MODE
DEFINE KEY=Ctrl/D ERASE WORD
DEFINE KEY=GOLD W START OF LINE
DEFINE KEY=KP5 FILL PARAGRAPH
!
!Binds the EDT forward function (KP4 on
!EDT keypad) to GOLD F
!
DEFINE KEY=GOLD F EDT KP4
```

A.9.4 Specifying Initialization Files

You can specify an initialization file with the /INITIALIZATION qualifier, defined as EVE$INIT in your LOGIN.COM file or named EVE$INIT.EVE in your SYS$LOGIN directory. The following command invokes EVE with the initialization file named MY_INIT:

```
$ EDIT/TPU/INIT=WORK1:[ALEXIS]MY_INIT
```

A.9.5 Default Initialization File

By default, DECTPU uses the initialization file whose logical name is EVE$INIT. If you define this logical name in your LOGIN.COM file, DECTPU automatically uses your initialization file when you invoke EVE. For example, you could insert the following command in your LOGIN.COM file:

```
$ DEFINE EVE$INIT WORK1:[ALEXIS]MY_INIT.EVE
```

When EVE starts up, it looks first for a section file, then for a command file, and finally for an initialization file. Because an initialization file is executed after a section file and a command file, the definitions in an initialization file override those in a section file or a command file. For this reason, place commands that define the editing environment in either your command file or your initialization file.

A.9.6 Commands That Define the Environment

Commands that define the environment include the following:

- SET CURSOR BOUND or FREE
- SET FIND WHITESPACE or NOWHITESPACE
- SET GOLD KEY
- SET KEYPAD
- SET LEFT MARGIN
- SET RIGHT MARGIN
- SET SCROLL MARGINS
- SET TABS AT or EVERY
- SET TABS SPACES, MOVEMENT, or INSERT
- SET TABS VISIBLE or INVISIBLE

- SET WIDTH
- SET WILDCARD VMS or ULTRIX
- SET WRAP or NOWRAP
- The default mode of the buffer: CHANGE MODE, OVERSTRIKE MODE, or INSERT MODE
- The default direction of the buffer: CHANGE DIRECTION, FORWARD, or REVERSE

A.10 Saving Your Customizations in Startup Files

A.10.1 Overview

You can save key definitions, learn sequences, and DECTPU procedures in a startup file. With a startup file, you can save all the modifications you have made to EVE so you do not have to recreate your modifications at each editing session.

A.10.2 Types of Startup Files

EVE has three types of startup files:

- Section files
- Command files
- Initialization files

You can customize section files and command files interactively from the EVE editor. You create initialization files separately.

When saving your customizations in a section file or command file, use the SAVE ATTRIBUTES command or the SAVE EXTENDED EVE command when you exit from or quit the editor.

A.10.3 EVE Commands for Saving Attributes

This list summarizes the commands for saving attributes:

- **SAVE ATTRIBUTES**

  Saves attribute settings and user menu definitions in a section file or command file, depending on your responses to EVE prompts or on settings done with other EVE commands. If you save in a section file, the effect is the same as SAVE EXTENDED EVE. If you save in a command file, EVE generates a specially marked block of DECTPU statements for attribute settings and menu definitions and either creates a command file or updates an existing command file with this block of statements.

- **SAVE SYSTEM ATTRIBUTES**

  Saves EVE default attribute settings in a section file or command file. This is useful if you want to restore your section file or command file to the standard EVE settings and menu definitions.

- **SAVE EXTENDED EVE**

  Creates a section file, saving attribute settings, key definitions, menu definitions, compiled procedures, and other extensions, such as global variables set with a DECTPU statement. If you do not specify a section file on the command line, EVE prompts you for one or uses your default section file (if you set a default).

- **SET DEFAULT COMMAND FILE**

  Determines the command file for saving attributes. Does not determine the command file to be executed at startup, if any.

- **SET DEFAULT SECTION FILE**

  Determines the section file for saving attributes. Does not determine the section file to be executed at startup.

- **SET EXIT ATTRIBUTE CHECK**

  If you changed attributes, then when you exit or quit, EVE asks if you want to save your changes. Default setting.

- **SET NODEFAULT COMMAND FILE**

  When you save attributes, the default command file is TPU$COMMAND.TPU in your current directory or the command file that was executed at startup. Default setting.

- **SET NODEFAULT SECTION FILE**

  When you save attributes, EVE asks for the name of the section file you want to create (unless you disabled section file prompting). Default setting.

- **SET NOEXIT ATTRIBUTE CHECK**

  Disables attribute checking, typically to speed up or simplify exiting or quitting. Does not apply to the editing session in which you issue the command, but only to the editing sessions in which you use the saved section file or command file.

- **SET NOSECTION FILE PROMPTING**

  Disables prompting for a section file when you save attributes, typically to speed up or simplify saving attributes in a default section file or in a command file.

- **SET SECTION FILE PROMPTING**

  When you save attributes, EVE prompts you for the name of a section file. Default setting.

A.11 Saving Attributes

A.11.1 Overview

If you have changed attributes and not saved them, EVE asks if you want to save the changed attributes when you exit from

A.11.2 Example: Saving Attributes

The following example shows how you can save attributes when you exit from EVE:

```
Command: SET CURSOR BOUND
Command: MOVING_TEXT
Command: SET TABS VISIBLE
           .
           .
           .
Command: EXIT
Attributes were changed.  Save them? [YES]
```

If you want to save the changes, press the Return key. EVE then executes breaka SAVE ATTRIBUTES command before exiting. If you do not want to save the changes, type No and press Return. EVE then continues exiting.

A.11.3 Disabling Prompting

To disable this prompting, for a faster or simpler exit, use the SET NOEXIT ATTRIBUTE CHECK command. However, the command does not apply to the current editing session because exit checking is itself a global setting and can be saved in a section file or command file. After you save it, the setting applies to future editing sessions in which you use the relevant section file or command file.

Other global attributes (such as scroll margins or the types of wildcards) and any buffer-specific attributes (such as margins or tab stops) are not saved in a section file or a command file. Typically, you use an initialization file for those settings.

A.11.4 Using Customizations

There are several ways in which you can use all your customizations in future editing sessions. You can combine different types of startup files in the following ways:

- Section file and an initialization file
- Command file (EVE-generated code) and an initialization file

  If you have limited disk space, use a command file instead of a section file.
- Command file (EVE-generated and user-generated code)

  You can write DECTPU procedures for attributes that typically are contained in an initialization file and use them in a command file containing EVE-generated code. Command files execute more quickly than an initialization file and offer more sophisticated editing tools.

A.12 Key Definitions in EVE Startup Files

This section lists the categories of commands for EVE attributes and features and shows in what type of startup file you can save them. By placing your definitions and procedures in a startup file, you can invoke the editor and automatically establish the editing environment your task requires.

A.12.1 Key Definitions

The following table lists key definitions and what type of files they can be used in:

Key Definitions	Section	Command	Initialization
DEFINE KEY	X	X	X
LEARN	X	–	–
SET FUNC KEYS [NO]DECWINDOWS	X	X	X
SET [NO]GOLD KEY	X	X	X
SET KEYPAD [NO]EDT	X	X	X
SET KEYPAD [NO]WPS	X	X	X
SET KEYPAD VT100	X	X	X
SET KEYPAD NUMERIC	X	X	X
UNDEFINE KEY	X	X	X

A.12.2 Global Settings-1

The following table lists global settings-1 and what type of files they can be used in:

Global Settings-1	Section	Command	Initialization
SET BOX [NO]PAD	X	X	X
SET BOX [NO]SELECT	X	X	X
SET CURSOR FREE or BOUND	X	X	X
SET [NO]CLIPBOARD	X	X	X
SET [NO]DEFAULT COMMAND FILE	X	X	X
SET [NO]DEFAULT SECTION FILE	X	X	X
SET [NO]EXIT ATTRIBUTE CHECK	X	X	X
SET FIND CASE [NO]EXACT	X	X	X
SET [NO]PENDING DELETE	X	X	X
SET [NO]SECTION FILE PROMPTING	X	X	X
SET TABS INSERT, MOVEMENT, or SPACES	X	X	X

Global Settings-1	Section	Command	Initialization
SET TABS [IN]VISIBLE	X	X	X

A.12.3 Global Settings-2

The following table lists global settings-2 and what type of files they can be used in:

Global Settings-2	Section	Command	Initialization
SET FIND [NO]WHITESPACE	–	–	X
SET SCROLL MARGINS	–	–	X
SET WIDTH	–	–	X
SET WILDCARDS VMS or ULTRIX	–	–	X

A.12.4 Buffer Settings

The following table lists buffer settings and what type of files they can be used in:

Buffer Settings	Section	Command	Initialization
FORWARD or REVERSE	–	–	X
INSERT MODE or OVERSTRIKE MODE	–	–	X
SET BUFFER	–	–	X
SET [NO]JOURNALING ALL	–	–	X
SET LEFT MARGIN	–	–	X
SET PARAGRAPH INDENT	–	–	X
SET RIGHT MARGIN	–	–	X
SET TABS AT or EVERY	–	–	X
SET [NO]WRAP	–	–	X

A.12.5 DECTPU Procedures

The following table lists DECTPU procedures and what type of files they can be used in:

DECTPU Procedures	Section	Command	Initialization
	X	X	–

A.12.6 Saving in a Section File

To save a section file, use the SAVE EXTENDED EVE command or the SAVE ATTRIBUTES command. Using SAVE EXTENDED EVE, you can specify the section file on the command line or let EVE prompt you for the section file name. Using SAVE ATTRIBUTES, you specify the section file as a response to a prompt.

To speed up saving in a section file, you can set a default section file—that is, the section file you want to save in without having to specify the file each time you save attributes—and you can disable section file prompting.

A.12.7 Example

The command shown in this example saves attributes and other customized settings in a section file entitled MYSEC.TPU$SECTION in the current directory:

```
Command: SAVE ATTRIBUTES
Save attributes in a section file [YES]? Return
File to save in: mysec
DISK$1:[USER]MYSEC.TPU$SECTION;1 created
```

A.12.8 EVE Settings for Saving Attributes

The following table shows the interaction of the settings for default section file and section file prompting:

Command Settings	Effect with SAVE ATTRIBUTES
SET DEFAULT SECTION FILE SET SECTION FILE PROMPTING	When you save attributes, EVE ask you whether to save in a section file. If you respond Yes (the default response), EVE saves in your default section file. If you respond No, EVE asks whether to save in a command file.
SET DEFAULT SECTION FILE SET NOSECTION FILE PROMPTING	When you save attributes, EVE saves in your default section file without prompting.
SET NODEFAULT SECTION FILE SET SECTION FILE PROMPTING	Default settings. When you save attributes, EVE asks whether to save in a section file. If you respond Yes, EVE asks for the name of a section file. If you respond No, EVE asks whether to save in a command file.
SET NODEFAULT SECTION FILE SET NOSECTION FILE PROMPTING	When you save attributes, EVE asks whether to save in a command file.

A.12.9 Specifying Section Files

Typically, when you use SET DEFAULT SECTION FILE, you specify the section file you are going to use at startup for future editing sessions. The command does not determine the section file to be executed when you invoke the editor, but only the section file in which you save attributes and other customized settings. To specify the section file you want executed at startup, do either of the following:

- Use EDIT/TPU/SECTION and specify the section file you want to use:

  ```
  $ EDIT/TPU/SECTION=MYEVE
  ```

- Define the logical name TPU$SECTION to specify the section file and then use the EDIT/TPU command:

```
$ DEFINE TPU$SECTION SYS$LOGIN:MYEVE
$ EDIT/TPU
```

In specifying the section file to be executed, you must use a complete file specification, including the device (or disk) and directory; otherwise, DECTPU assumes the section file is in SYS$SHARE.

Section files may be quite large, depending on the number of customized settings you save. If you have limited disk space, you should save in a command file, which requires less disk space. For more information about creating and using section files, see the EVE online help topic called Section Files.

A.12.10 Saving in a Command File

When you use the SAVE ATTRIBUTES command or when you save attributes on exiting or quitting, you can have EVE create or update a command file. To save attributes in a command file, use the SAVE ATTRIBUTES command, as follows:

```
Command: SAVE ATTRIBUTES
Save attributes in a section file [YES]? no
Save attributes in a command file [YES]? Return
Enter file name [TPU$COMMAND.TPU] MYCOM
14 written to file DISK$1:[USER]MYCOM.TPU;1
```

The prompt for the command file name shows, in brackets, the default command file that EVE uses if you press the Return key at the prompt without typing a file name. This default is one of the following:

- The command file specified with the /COMMAND qualifier when you invoked EVE
- The command file defined by the logical name TPU$COMMAND
- A command file called TPU$COMMAND.TPU in your current directory

A.12.11 Setting a Default Command File

You can set your preferred default command file—that is, the command file you want EVE to create or update, without having to specify the file each time you save attributes. For example, the following command sets your default command file as MYCOM.TPU in your current directory:

```
Command: SET DEFAULT COMMAND FILE MYCOM
```

If you want to save in a command file rather than in a section file, you should also use the SET NOSECTION FILE PROMPTING command. Then, when you save attributes, EVE asks whether to save in a command file without first asking whether to save in a section file.

A.12.12 Specifying Startup Command Files

When you use SET DEFAULT COMMAND FILE, you usually specify the command file you are going to use at startup for future editing sessions. The command does not determine the command file to be executed when you invoke EVE, but only the command file in which you save attributes and menu definitions. To specify the command file you want executed at startup, do any of the following:

- Use EDIT/TPU/COMMAND and specify the command file you want to use:

  ```
  $ EDIT/TPU/COMMAND=MYEVE
  ```

- Define the logical name TPU$COMMAND to specify the command file and then invoke EVE using EDIT/TPU:

  ```
  $ DEFINE TPU$COMMAND SYS$LOGIN:MYEVE
  $ EDIT/TPU
  ```

- Name the command file TPU$COMMAND.TPU in your current directory and then use EDIT/TPU to invoke EVE.

For more information about creating and using command files, see the EVE online help topic entitled Command Files.

A.13 Converting from EDT to EVE

A.13.1 Overview

If you are accustomed to the EDT editor, you can customize EVE to work in similar ways by using a section file, an initialization file, or both, or by using DECTPU procedures.

Typically, you save key definitions, learn sequences, and other extensions in a section file (created with the SAVE EXTENDED EVE command). Use an EVE initialization file to set editing preferences or private defaults, such as margins and tabs, which are not saved in the section file.

A.13.2 Using the SET KEYPAD EDT Command

The SET KEYPAD EDT command defines several keys to emulate EDT. You can put the command in your EVE initialization file or save the keypad setting in a section file. Most keypad functions work as in EDT, although the names may differ. For more information, see the online help topic called EDT Differences.

A.13.3 Defining Keys for EVE Commands

Use DEFINE KEY commands to define keys that are not otherwise defined by SET KEYPAD EDT. Put the key-definition commands in your initialization file or save the definitions in a section file.

A.13.4 Example: Defining Keys for EVE Commands

The following sets of EDT and EVE key definitions are equivalent:

In EDT:

```
DEF KEY  GOLD 2    "EXT SHOW BUFFER."
DEF KEY  GOLD l    "CHGLW."
DEF KEY  GOLD u    "CHGUW."
DEF KEY  GOLD 10   "EXT FIND=?.."
DEF KEY  GOLD 9    "CUTSR PASTE."
DEF KEY  cont n    "EXT QUIT."
DEF KEY  func 34   "SHL."
```

In EVE:

```
DEF KEY=  GOLD E2   SHOW BUFFERS
DEF KEY=  GOLD l    LOWERCASE WORD
DEF KEY=  GOLD U    UPPERCASE WORD
DEF KEY=  GOLD PF2  BUFFER
DEF KEY=  GOLD KP9  STORE TEXT
DEF KEY=  Ctrl/N    QUIT
DEF KEY=  F20       SHIFT RIGHT 8
```

Note the differences between EDT and EVE in some key names, as well as differences in command names. For more information about key names, see Section A.2.

A.13.5 Setting Bound Cursor Motion

Use the SET CURSOR BOUND command to enable an EDT style bound cursor. By default, EVE uses a free cursor, which you can move anywhere in the buffer. You can save the setting in your section file or DECTPU command file.

A.13.6 Setting the Right Margin for Wrapping Text

Put the SET RIGHT MARGIN command in your EVE initialization file to set a wrap limit for entering text and for FILL commands. For example, the following EDT and EVE commands are equivalent:

In EDT:

```
SET WRAP 70
```

In EVE:

```
SET RIGHT MARGIN 70
```

(The EVE command SET WRAP corresponds to the EDT command SET NOTRUNCATE.)

A.13.7 Setting Scroll Margins for Moving the Cursor

Put the SET SCROLL MARGINS command in your EVE initialization file to set distances for scrolling to begin automatically as you move the cursor up or down. For example, with a 24-line terminal screen (21-line main window), the following EDT and EVE commands are equivalent:

In EDT:

```
SET CURSOR 5:15
```

In EVE:

```
SET SCROLL MARGINS 5 6
```

EVE scroll margins are measured from the top and bottom, respectively. In EDT, both scroll margins are measured from the top. You can specify numbers of lines or percentages of the window size. Also, the size of the EVE main window depends on your terminal settings. For example, on a workstation, the EVE main window may be longer than 21 lines.

A.13.8 Setting Searches for Exact Case

Searches follow EVE rules for case sensitivity. Put SET FIND commands in your EVE initialization file to set the way you want searches to work. For example, the following EDT and EVE commands are nearly equivalent:

In EDT:

```
SET SEARCH EXACT
```

In EVE:

```
SET FIND CASE EXACT
```

These commands are not exact equivalences because EVE always matches diacritical marks exactly as entered in the search string.

A.13.9 Converting EDT Macros to DECTPU Procedures

Use DECTPU procedures in place of EDT macros. Create a buffer that contains the procedures and then use EXTEND commands to compile the procedures with EXTEND commands. Or, put the procedures in a DECTPU command file and then use the /COMMAND qualifier to invoke EVE. In either case, you can save the compiled procedures in your section file.

You can also use the LEARN command to bind the corresponding EVE commands to a single key. You can then save the key definition in your section file. Another method is to put the corresponding EVE commands in an initialization file.

A.13.10 Examples: Converting EDT Macros to DECTPU Procedures

The following examples show a macro from an EDT startup file which is translated into a DECTPU procedure. Each macro creates a new command, WIDEN, which sets the display to 132 columns and sets the right margin to 120.

EDT Macro:

```
FIND =WIDEN
INSERT;SET SCREEN 132
INSERT;SET WRAP 120
FIND =MAIN.
```

DECTPU Procedure:

```
PROCEDURE EVE_WIDEN;
   EVE_SET_WIDTH (132);
   EVE_SET_RIGHT_MARGIN (120);
ENDPROCEDURE;
```

To execute the macro or procedure, use the following commands:

In EDT:

```
* DEFINE MACRO WIDEN
* WIDEN
```

In EVE:

```
Command: EXTEND EVE WIDEN
Command: WIDEN
```

A.13.11 Converting EDT Nokeypad Statements to DECTPU Procedures

You can usually convert EDT macros and key definitions that use nokeypad specifiers into DECTPU procedures or learn sequences. The following examples show an EDT key definition that uses nokeypad mode and the corresponding DECTPU procedure and key definition. In each case, you define Comma on the numeric keypad to transpose or swap the current and previous character. The –C in EDT nokeypad statements can be translated as MOVE_HORIZONTAL (–1) in DECTPU procedures.

In EDT:

```
DEFINE KEY 19 AS "-C D1C +C UNDC."
```

In DECTPU:

```
PROCEDURE USER_TRANSPOSE
   LOCAL  SWAP_THIS;

   SWAP_THIS  :=  ERASE_CHARACTER (1);

   MOVE_HORIZONTAL (-1);
   EVE$INSERT_TEXT (SWAP_THIS);
   RETURN (TRUE);
ENDPROCEDURE;

EVE$DEFINE_KEY ("USER_TRANSPOSE", COMMA, , EVE$X_USER_KEYS);
```

A.13.12 Using the WPS Keypad Ruler Key

Setting the EDT keypad does not define keys for EDT style tab adjustment. However, you can get similar effects by defining a key for the WPS keypad Ruler key (GOLD R) and then using the ruler to add or delete tab stops.

A.13.13 Tutorial: Using the WPS Keypad Ruler

In this tutorial, the command defines F20 as the WPS Ruler key (without having to enable the WPS keypad):

```
Command: DEFINE KEY= F20  WPS GOLD-R
```

Then, to add or delete tab stops, follow these steps:

Step	Task
1	Press whatever key you have defined as the Ruler key. EVE displays a ruler at the bottom of the current window (just above the status line for the window). The cursor appears in the ruler. Tab stops are marked with a *T*.
2	Put the cursor where you want to add or delete a tab stop. For example, press the left and right arrow keys to move to a particular column in the ruler or press the Tab key to move to the next tab stop (*T*) in the ruler.
3	Type a *T* or *t* at that location to set the tab stop or, if there is already a tab there, to delete it. The new tab stops are immediately applied to the buffer you were editing.
4	Repeat steps 2 and 3 to add or delete other tab stops.
5	To exit from the ruler and resume editing, press the Return key or GOLD Return.

For more information about the WPS Ruler key, including a list of the keys for moving the cursor in the ruler, see the description of the SET KEYPAD WPS command in the *Guide to the Extensible Versatile Editor*.

B

Character Sets

B.1 Overview

This appendix describes the DEC Multinational character set and the DCL character set. It includes information about:

- DEC Multinational character set
- DCL character set

B.2 DEC Multinational Character Set

B.2.1 Overview

The DEC Multinational character set is an 8-bit character set with 256 characters; the first 128 characters in the set correspond to the ASCII character set. Each character has a value in the range 0 to 255 decimal.

B.2.2 ASCII Character Set

Section B.2.4 represents the ASCII character set (characters with decimal values 0 through 127). The first half of each numbered column identifies the character as you would enter it on a VT-series terminal or workstation or as you would see it on a printer (except for the nonprintable characters). The remaining half of each column identifies the character by the binary value of the byte; the value is stated in three radixes—octal, decimal, and hexadecimal.

B.2.3 Example

The uppercase letter A has, under ASCII conventions, a storage value of hexadecimal 41 (a bit configuration of 01000001), equivalent to 101 in octal notation and 65 in decimal notation. DCL uses hexadecimal values to perform string comparisons. (For a description of string comparisons, see Chapter 14.)

B.2.4 ASCII Character Set, Part 1

The following figure represents the first half of the DEC Multinational Character set:

Row	b8 b7 b6 b5 b4 b3 b2 b1	0	1	2	3	4	5	6	7
0	0 0 0 0	NUL 0 0 0	DLE 20 16 10	SP 40 32 20	0 60 48 30	@ 100 64 40	P 120 80 50	` 140 96 60	p 160 112 70
1	0 0 0 1	SOH 1 1 1	DC1 (XON) 21 17 11	! 41 33 21	1 61 49 31	A 101 65 41	Q 121 81 51	a 141 97 61	q 161 113 71
2	0 0 1 0	STX 2 2 2	DC2 22 18 12	" 42 34 22	2 62 50 32	B 102 66 42	R 122 82 52	b 142 98 62	r 162 114 72
3	0 0 1 1	ETX 3 3 3	DC3 (XOFF) 23 19 13	# 43 35 23	3 63 51 33	C 103 67 43	S 123 83 53	c 143 99 63	s 163 115 73
4	0 1 0 0	EOT 4 4 4	DC4 24 20 14	$ 44 36 24	4 64 52 34	D 104 68 44	T 124 84 54	d 144 100 64	t 164 116 74
5	0 1 0 1	ENQ 5 5 5	NAK 25 21 15	% 45 37 25	5 65 53 35	E 105 69 45	U 125 85 55	e 145 101 65	u 165 117 75
6	0 1 1 0	ACK 6 6 6	SYN 26 22 16	& 46 38 26	6 66 54 36	F 106 70 46	V 126 86 56	f 146 102 66	v 166 118 76
7	0 1 1 1	BEL 7 7 7	ETB 27 23 17	´ 47 39 27	7 67 55 37	G 107 71 47	W 127 87 57	g 147 103 67	w 167 119 77
8	1 0 0 0	BS 10 8 8	CAN 30 24 18	(50 40 28	8 70 56 38	H 110 72 48	X 130 88 58	h 150 104 68	x 170 120 78
9	1 0 0 1	HT 11 9 9	EM 31 25 19	) 51 41 29	9 71 57 39	I 111 73 49	Y 131 89 59	i 151 105 69	y 171 121 79
10	1 0 1 0	LF 12 10 A	SUB 32 26 1A	* 52 42 2A	: 72 58 3A	J 112 74 4A	Z 132 90 5A	j 152 106 6A	z 172 122 7A
11	1 0 1 1	VT 13 11 B	ESC 33 27 1B	+ 53 43 2B	; 73 59 3B	K 113 75 4B	[133 91 5B	k 153 107 6B	{ 173 123 7B
12	1 1 0 0	FF 14 12 C	FS 34 28 1C	, 54 44 2C	< 74 60 3C	L 114 76 4C	\ 134 92 5C	l 154 108 6C	| 174 124 7C
13	1 1 0 1	CR 15 13 D	GS 35 29 1D	– 56 45 2D	= 75 61 3D	M 115 77 4D	] 135 93 5D	m 155 109 6D	} 175 125 7D
14	1 1 1 0	SO 16 14 E	RS 36 30 1E	. 56 46 2E	> 76 62 3E	N 116 78 4E	^ 136 94 5E	n 156 110 6E	~ 176 126 7E
15	1 1 1 1	SI 17 15 F	US 37 31 1F	/ 57 47 2F	? 77 63 3F	O 117 79 4F	_ 137 95 5F	o 157 111 6F	DEL 177 127 7F

Key

Character: ESC | 33 (Octal) 27 (Decimal) 1B (Hex)

ZK–1752–GE

B.2.5 ASCII Character Set, Part 2

The following figure represents the second half of the DEC Multinational character set (characters with decimal values 128 through 255). The first half of each numbered column identifies the character as it appears on a VT200 or VT300 series terminal or printer (you cannot display these characters on a VT100 series terminal).

8		9		10		11		12		13		14		15		Column	
1 0 0 0		1 0 0 1		1 0 1 0		1 0 1 1		1 1 0 0		1 1 0 1		1 1 1 0		1 1 1 1		b8 **Bits** b7 b6 b5 b4 b3 b2 b1	Row
	200 128 80	**DCS**	220 144 90		240 160 A0	°	260 176 B0	**À**	300 192 C0		320 208 D0	**à**	340 224 E0		360 240 F0	0 0 0 0	**0**
	201 129 81	**PU1**	221 145 91	¡	241 161 A1	±	261 177 B1	**Á**	301 193 C1	**Ñ**	321 209 D1	**á**	341 225 E1	**ñ**	361 241 F1	0 0 0 1	**1**
	202 130 82	**PU2**	222 146 92	¢	242 162 A2	²	262 178 B2	**Â**	302 194 C2	**Ò**	322 210 D2	**â**	342 226 E2	**ò**	362 242 F2	0 0 1 0	**2**
	203 131 83	**STS**	223 147 93	£	243 163 A3	³	263 179 B3	**Ã**	303 195 C3	**Ó**	323 211 D3	**ã**	343 227 E3	**ó**	363 243 F3	0 0 1 1	**3**
IND	204 132 84	**CCH**	224 148 94		244 164 A4		264 180 B4	**Ä**	304 196 C4	**Ô**	324 212 D4	**ä**	344 228 E4	**ô**	364 244 F4	0 1 0 0	**4**
NEL	205 133 85	**MW**	225 149 95	¥	245 165 A5	µ	265 181 B5	**Å**	305 197 C5	**Õ**	325 213 D5	**å**	345 229 E5	**õ**	365 245 F5	0 1 0 1	**5**
SSA	206 134 86	**SPA**	226 150 96		246 166 A6	¶	266 182 B6	**Æ**	306 198 C6	**Ö**	326 214 D6	**æ**	346 230 E6	**ö**	366 246 F6	0 1 1 0	**6**
ESA	207 135 87	**EPA**	227 151 97	§	247 167 A7	·	267 183 B7	**Ç**	307 199 C7	**Œ**	327 215 D7	**ç**	347 231 E7	**œ**	367 247 F7	0 1 1 1	**7**
HTS	210 136 88		230 152 98	¤	250 168 A8		270 184 B8	**È**	310 200 C8	**Ø**	330 216 D8	**è**	350 232 E8	**ø**	370 248 F8	1 0 0 0	**8**
HTJ	211 137 89		231 153 99	©	251 169 A9	¹	271 185 B9	**É**	311 201 C9	**Ù**	331 217 D9	**é**	351 233 E9	**ù**	371 249 F9	1 0 0 1	**9**
VTS	212 138 8A		232 154 9A	ª	252 170 AA	º	272 186 BA	**Ê**	312 302 CA	**Ú**	332 218 DA	**ê**	352 234 EA	**ú**	372 250 FA	1 0 1 0	**10**
PLD	213 139 8B	**CSI**	233 155 9B	«	253 171 AB	»	273 187 BB	**Ë**	313 203 CB	**Û**	333 219 DB	**ë**	353 235 EB	**û**	373 251 FB	1 0 1 1	**11**
PLU	214 140 8C	**ST**	234 156 9C		254 172 AC	¼	274 188 BC	**Ì**	314 204 CC	**Ü**	334 220 DC	**ì**	354 236 EC	**ü**	374 252 FC	1 1 0 0	**12**
RI	215 141 8D	**OSC**	235 157 9D		255 173 AD	½	275 189 BD	**Í**	315 205 CD	**Ÿ**	335 221 DD	**í**	355 237 ED	**ÿ**	375 253 FD	1 1 0 1	**13**
SS2	216 142 8E	**PM**	236 158 9E		256 174 AE		276 190 BE	**Î**	316 206 CE		336 222 DE	**î**	356 238 EE		376 254 FE	1 1 1 0	**14**
SS3	217 143 8F	**APC**	237 159 9F		257 175 AF	¿	277 191 BF	**Ï**	317 207 CF	**ß**	337 223 DF	**ï**	357 239 EF		377 255 FF	1 1 1 1	**15**

Key

Character

ESC	33 27 1B

Octal
Decimal
Hex

ZK–1753–GE

B.3 DCL Character Set

Table B–1 lists the characters in the DCL character set:

Table B–1 DCL Character Set

Symbol	Name	Meaning
@	At sign	Places the contents of a command procedure file in the command input stream.
:	Colon	Device name delimiter in a file specification. A double colon (::) is a node name delimiter. A colon also acts as a qualifier delimiter. It separates a qualifier name from its value.
/	Slash	Qualifier prefix.
+	Plus sign	Parameter separator. With some commands it acts as a parameter concatenator. The plus sign is also recognized as a string concatenation operator, a unary plus sign, and an addition operator in a numeric expression.
,	Comma	List element separator for parameters or argument lists.
-	Hyphen	Continuation character. The hyphen is also recognized as a string reduction operator, a unary minus sign, a subtraction operator in a numeric expression, and a directory-searching wildcard character.
( )	Parentheses	List delimiters for argument list. Parentheses are also used to indicate the order of operations in a numeric expression.
[ ]	Square brackets	Directory name delimiters in a file specification. Equivalent to angle brackets.
<>	Angle brackets	Directory name delimiters in a file specification. Equivalent to square brackets.
?	Question mark	Help character.
&	Ampersand	Execution-time substitution operator. Otherwise, a reserved special character.
\	Backslash	Reserved special character.

(continued on next page)

Table B–1 (Cont.) DCL Character Set

Symbol	Name	Meaning
=	Equal sign	Qualifier value delimiter. It separates a qualifier name from its argument. The equal sign (=) can also be used as an assignment statement when defining symbols.
^	Circumflex	Reserved special character.
#	Number sign	Reserved special character.
*	Asterisk	Wildcard character in a file specification. The asterisk is also used as a multiplication operator in a numeric expression and as an abbreviation delimiter in a symbol definition.
'	Apostrophe	Substitution operator.
.	Period	File type and version number delimiter in a file specification. Also used as a subdirectory delimiter.
;	Semicolon	Version number delimiter in a file specification.
%	Percent sign	Wildcard character in a file specification. Also used as a radix operator.
!	Exclamation point	Indicates a comment.
"	Quotation mark	Literal string delimiter.

C
Annotated Command Procedures

C.1 Overview

This appendix contains complete command procedures that demonstrate the concepts and techniques discussed in Chapter 15, Chapter 16 and Chapter 17. Each section in this appendix discusses one command procedure and contains the following information:

- The name of the procedure
- A listing of the procedure
- Notes that explain concepts or techniques used by the procedure
- The results of a sample execution of the procedure

C.2 CONVERT.COM Command Procedure

C.2.1 Introduction

This command procedure converts an absolute time (for a time in the future) to a delta time and determines the time between the current time and the time that you specify. The procedure illustrates the use of the F$TIME and F$CVTIME lexical functions and the use of assignment statements to perform arithmetic calculations and to concatenate symbol values.

C.2.2 Example: CONVERT.COM

```
$ ! Procedure to convert an absolute time to a delta time.
$ ! The delta time is returned as the global symbol WAIT_TIME.
$ ! P1 is the time to be converted.
$ ! P2 is an optional parameter - SHOW - that causes the
$ ! procedure to display WAIT_TIME before exiting
$ !
$ ! Check for inquiry
$ !
$ IF P1 .EQS. "?" .OR. P1 .EQS. "" THEN GOTO TELL          ❶
$ !
$ ! Verify the parameter:  hours must be less than 24
$ !                       minutes must be less than 60
$ !                       time string must contain only hours
$ !                       and minutes
$ !
$ ! Change error and message handling to
$ ! use message at BADTIME
```

```
$ !
$ ON WARNING THEN GOTO BADTIME                                    ❷
$ SAVE_MESSAGE = F$ENVIRONMENT("MESSAGE")
$ SET MESSAGE/NOFACILITY/NOIDENTIFICATION/NOSEVERITY/NOTEXT
$ TEMP = F$CVTIME(P1)
$ !
$ ! Restore default error handling and message format
$ ON ERROR THEN EXIT
$ SET MESSAGE'SAVE_MESSAGE'
$ !
$ IF F$LENGTH(P1) .NE. 5 .OR. -                                   ❸
       F$LOCATE(":",P1) .NE. 2 -
       THEN GOTO BADTIME
$ !
$ ! Get the current time
$ !
$ TIME = F$TIME()                                                 ❹
$ !
$ ! Extract the hour and minute fields from both the current time
$ ! value (TIME) and the future time (P1)
$ !
$ MINUTES = F$CVTIME(TIME,"ABSOLUTE","MINUTE")       ! Current minutes ❺
$ HOURS = F$CVTIME(TIME,"ABSOLUTE","HOUR")           ! Current hours
$ FUTURE_MINUTES = F$CVTIME(P1,"ABSOLUTE","MINUTE")  ! Minutes in future time
$ FUTURE_HOURS = F$CVTIME(P1,"ABSOLUTE","HOUR")      ! Hours in future time
$ !
$ !
$ ! Convert both time values to minutes
$ ! Note the implicit string to integer conversion being performed
$ !
$ CURRENT_TIME = HOURS*60 + MINUTES                               ❻
$ FUTURE_TIME = FUTURE_HOURS*60 + FUTURE_MINUTES
$ !
$ ! Compute difference between the future time and the current time
$ ! (in minutes)
$ !
$ !
$ MINUTES_TO_WAIT = FUTURE_TIME - CURRENT_TIME                    ❼
$ !
$ ! If the result is less than 0 the specified time is assumed to be
$ ! for the next day; more calculation is required.
$ !
$ IF MINUTES_TO_WAIT .LT. 0 THEN -                                ❽
       MINUTES_TO_WAIT = 24*60 + FUTURE_TIME - CURRENT_TIME
$ !
$ ! Start looping to determine the value in hours and minutes from
$ ! the value expressed all in minutes
$ !
$       HOURS_TO_WAIT = 0
$ HOURS_TO_WAIT_LOOP:                                             ❾
$       IF MINUTES_TO_WAIT .LT. 60 THEN GOTO FINISH_COMPUTE
$       MINUTES_TO_WAIT = MINUTES_TO_WAIT - 60
$       HOURS_TO_WAIT = HOURS_TO_WAIT + 1
$       GOTO HOURS_TO_WAIT_LOOP
$ FINISH_COMPUTE:
```

```
$ !
$ ! Construct the delta time string in the proper format
$ !
$ WAIT_TIME == F$STRING(HOURS_TO_WAIT)+ ":" + F$STRING(MINUTES_TO_WAIT)- ❿
  + ":00.00"
$ !
$ ! Examine the second parameter
$ !
$ IF P2 .EQS.  "SHOW" THEN SHOW SYMBOL WAIT_TIME          ⓫
$ !
$ ! Normal exit
$ !
$ EXIT
$ !
$ BADTIME:                                                ⓬
$ ! Exit taken if first parameter is not formatted correctly
$ ! EXIT command returns but does not display error status
$ !
$ SET MESSAGE'SAVE_MESSAGE'
$ WRITE SYS$OUTPUT  "Invalid time value: ",P1,", format must be hh:mm"
$ WRITE SYS$OUTPUT "Hours must be less than 24; minutes must be less than 60"
$ EXIT %X10000000
$ !
$ !
$ TELL:                                                   ⓭
$ ! Display message and exit if user enters inquiry or enters
$ ! an illegal parameter
$ !
$ TYPE SYS$INPUT
        This procedure converts an absolute time value to
        a delta time value.  The absolute time must be in
        the form hh:mm and must indicate a time in the future.
        On return, the global symbol WAIT_TIME contains the
        converted time value.  If you enter the keyword SHOW
        as the second parameter, the procedure displays the
        resulting value in the output stream. To invoke this
        procedure, use the following syntax:

                      @CONVERT hh:mm [SHOW]
$ EXIT
```

C.2.3 Notes for CONVERT.COM Command Procedure

❶ The procedure checks whether the parameter was omitted or whether the value entered for a parameter is the question mark (?) character. In either case, the procedure branches to the label TELL.

❷ The procedure uses the F$CVTIME function to verify that the time value is a valid 24-hour clock time; the F$CVTIME returns a warning message if the input time is not valid. If the F$CVTIME function returns an error, the procedure changes the default ON action to direct control to the label BADTIME.

The procedure uses the F$ENVIRONMENT function to save the current message setting. It then sets the message format so that no warning or error messages are displayed. After

checking the time values, the procedure restores the default ON condition and message format.

❸ The procedure checks the format of the parameter. It must be a time value in the following format:

hh:mm

The IF command checks (1) that the length of the entered value is 5 characters and (2) that the third character (offset of 2) is a colon. The IF command contains the logical OR operator: if either expression is true (that is, if the length is not 5 or if there is not a colon in the third character position), the procedure branches to the label BADTIME.

❹ The F$TIME lexical function places the current time value in the symbol TIME.

❺ The F$CVTIME function extracts the "minute" and "hour" fields from the current time (saved in the symbol TIME). Then the F$CVTIME function extracts the "minute" and "hour") fields from the time you want to convert.

❻ These assignment statements convert the current and future times to minutes. When you use the symbols MINUTES, HOURS, FUTURE_HOURS, and FUTURE_MINUTES in the assignment statements, the system automatically converts these values to integers.

❼ The procedure then subtracts the current time (in minutes) from the future time (in minutes).

❽ If the result is less than 0, the future time is interpreted as being on the next day. In this case, the procedure adds 24 hours to the future time and then subtracts the current time.

❾ The procedure enters a loop in which it calculates, from the value of MINUTES_TO_WAIT, the number of hours. Each time through the loop, it checks whether MINUTES_TO_WAIT is greater than 60. If it is, the procedure subtracts 60 from MINUTES_TO_WAIT and adds 1 to the accumulator for the number of hours (HOURS_TO_WAIT).

❿ When the procedure exits from the loop, it concatenates the hours and minutes values into a time string. The symbols HOURS_TO_WAIT and MINUTES_TO_WAIT are replaced by their character string equivalents and separated with an intervening colon. The resulting string is assigned to the symbol WAIT_TIME, which holds the delta time value for the future time. WAIT_TIME is defined as a global symbol so that it is not deleted when the procedure CONVERT.COM exits.

⓫ If a second parameter, SHOW, was entered, the procedure displays the resulting time value. Otherwise, it exits.

⓬ At the label BADTIME, the procedure displays an error message that shows the incorrect value entered as well as the format it requires. After issuing the error message, CONVERT.COM exits. The EXIT command returns an error status in which the high-order digit is set to 1. This suppresses the display of an error message.

The procedure explicitly specifies an error status with the EXIT command, so you can execute CONVERT.COM from within another procedure. When CONVERT.COM completes, the calling procedure can determine whether a time was successfully translated.

⓭ At the label TELL, the procedure displays information about what the procedure does. The TYPE command displays the lines listed in SYS$INPUT, the input data stream.

C.2.4 Sample Execution for CONVERT.COM Command Procedure

```
$ SHOW TIME
10-JUN-1995 10:38:26
$ @CONVERT 12:00 SHOW
   WAIT_TIME = "1:22:00.00"
```

The SHOW TIME command displays the current date and time. CONVERT.COM is executed with the parameters 12:00 and SHOW. The procedure converts the absolute time 12:00 to a delta time value and displays it on the terminal.

C.3 REMINDER.COM Command Procedure

C.3.1 Overview

This command procedure displays a reminder message on your terminal at a specified time. The procedure prompts for the time you want the message to be displayed and for the text of the message. The procedure uses CONVERT.COM to convert the time to a delta time. The procedure then spawns a subprocess that waits until the specified time and displays your reminder message. The procedure illustrates the use of the F$ENVIRONMENT, F$VERIFY, and F$GETDVI functions.

C.3.2 Example: REMINDER.COM

```
$ ! Procedure to obtain a reminder message and display this
$ ! message on your terminal at the time you specify.
$ !
$ ! Save current states for procedure and image verification
$ ! Turn verification off for duration of procedure
$
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")                    ❶
$ SAVE_VERIFY_PROC = F$VERIFY(0)
$ !
$ ! Places the current process in a wait state until a specified
$ ! absolute time.  Then, it rings the bell on the terminal and
$ ! displays a message.
```

```
$ !
$ ! Prompt for absolute time
$ !
$
$ GET_TIME:
$ INQUIRE REMINDER_TIME  "Enter time to send reminder (hh:mm)"    ❷
$ INQUIRE MESSAGE_TEXT "Enter message"
$ !
$ ! Call the CONVERT.COM procedure to convert the absolute time
$ ! to a delta time
$ !
$ @DISK2:[JONES.TOOLS]CONVERT 'REMINDER_TIME'                     ❸
$ IF .NOT. $STATUS THEN GOTO BADTIME
$ !
$ !
$ ! Create a command file that will be executed
$ ! in a subprocess.  The subprocess will wait until
$ ! the specified time and then display your message
$ ! at the terminal.  If you are working at a DEC_CRT
$ ! terminal, the message has double size blinking
$ ! characters. Otherwise, the message has normal letters.
$ ! In either case, the terminal bell rings when the
$ ! message is displayed.
$
$ CREATE WAKEUP.COM                                               ❹
$ DECK                          ! Lines starting with $ are data lines
$ WAIT 'WAIT_TIME'                                                ❺
$ BELL[0,7] = %X07              ! Create symbol to ring the bell
$ IF F$GETDVI("SYS$OUTPUT","TT_DECCRT") .NES. "TRUE" THEN GOTO OTHER_TERM
$ !
$ DEC_CRT_ONLY:
$ ! Create symbols to set special graphics (for DEC_CRT terminals only)
$ !
$     SET_FLASH = "<ESC>[1;5m"    ! Turn on blinking characters
$     SET_NOFLASH = "<ESC>[0m"    ! Turn off blinking characters
$     TOP = "<ESC>#3"             ! Double size characters (top portion)
$     BOT = "<ESC>#4"             ! Double size characters (bottom portion)
$ !
$ ! Write double size, blinking message to the terminal and ring the bell
$ !
$     WRITE SYS$OUTPUT BELL, SET_FLASH, TOP, MESSAGE_TEXT
$     WRITE SYS$OUTPUT BELL, BOT, MESSAGE_TEXT
$     WRITE SYS$OUTPUT F$TIME(), SET_NOFLASH
$     GOTO CLEAN_UP
$ !
$ OTHER_TERM:
$      WRITE SYS$OUTPUT BELL,MESSAGE_TEXT
$      WRITE SYS$OUTPUT F$TIME()
$ !
$ CLEAN_UP:
$      DELETE WAKEUP.COM;*
$ EOD
```

```
$ !
$ ! Now continue executing commands.
$ !
$ SPAWN/NOWAIT/INPUT=WAKEUP.COM                                    ❻
$ END:                                                             ❼
$ !   Restore verification
$     SAVE_VERIFY_PROC = F$VERIFY(SAVE_VERIFY_PROC, SAVE_VERIFY_IMAGE)
$     EXIT
$ !
$ BADTIME:
$    WRITE SYS$OUTPUT "Time must be entered as hh:mm"
$    GOTO GET_TIME
```

C.3.3 Notes for REMINDER.COM Command Procedure

❶ The procedure uses the F$ENVIRONMENT function to save the image verification setting in the symbol SAVE_VERIFY_IMAGE. Next, the procedure uses the F$VERIFY function to save the procedure verification setting in the symbol SAVE_VERIFY_PROC. The F$VERIFY function also turns off both types of verification.

❷ The procedure uses the INQUIRE command to prompt for the time when the reminder message should be sent. This value is used as input to the procedure CONVERT.COM. The procedure also prompts for the text of the message.

❸ The procedure executes a nested procedure, CONVERT.COM. Be sure to specify the disk and directory as part of the file specification; this ensures that the system can locate CONVERT.COM regardless of the directory from which you execute REMINDER.COM.

CONVERT.COM converts your reminder to a delta time, and returns this time in the global symbol WAIT_TIME. This delta time indicates the time interval from the current time until the time when the message should be sent. If CONVERT.COM returns an error, the procedure branches to the label BADTIME.

❹ The procedure uses the CREATE command to create a new procedure, WAKEUP.COM. This procedure is executed from within a subprocess. To allow the CREATE command to read lines that begin with dollar signs, use the DECK and EOD commands to surround the input for the CREATE command. Therefore, all lines between the DECK and EOD commands are written to WAKEUP.COM.

❺ WAKEUP.COM performs the following tasks:

- It waits until the time indicated by the symbol WAIT_TIME.
- It creates the symbol BELL to ring the terminal bell.

- It determines whether the terminal is a DEC_CRT terminal and can accept escape sequences to display double-size, blinking characters. (To see whether you have a DEC_CRT terminal, enter the SHOW TERMINAL command and see whether this characteristic is listed.)
- If the terminal is a DEC_CRT terminal, then the procedure defines the symbols SET_FLASH, TOP, and BOT. These symbols cause the terminal to use flashing, double-size characters. The procedure also defines the symbol SET_NOFLASH to return the terminal to its normal state. To enter the escape character (<ESC>) when you create these definitions using the EDT editor, press the ESC key twice.

  After defining these symbols, the procedure writes three lines to the terminal. The first line rings the bell, turns on flashing characters, and displays (using double-size characters) the top half of your message. The second line rings the bell again, and displays the bottom half of your message. The third line writes the current time and then turns off the flash characteristic to return your terminal to normal.

  If you do not have a DEC_CRT terminal, then the procedure rings your terminal bell, and displays your message and the time.
- The DELETE command causes the procedure WAKEUP.COM to delete itself after it executes.

❻ After creating WAKEUP.COM, the procedure spawns a subprocess and directs the subprocess to use WAKEUP.COM as the input command file. The /NOWAIT qualifier allows you to continue working at your terminal while the subprocess executes commands from WAKEUP.COM. At the specified time, WAKEUP.COM displays your message on your terminal.

Note that, by default, the SPAWN command passes global and local symbols to a subprocess. Therefore, although you provide values for the symbols WAIT_TIME and MESSAGE_TEXT in REMINDER.COM, WAKEUP.COM can also access these symbols.

❼ The procedure restores the original verification settings before it exits.

C.3.4 Sample Execution for REMINDER.COM Command Procedure

```
$ @REMINDER
Enter time to send reminder (hh:mm): 12:00
Enter message: TIME FOR LUNCH
%DCL-S-SPAWNED, process BLUTO_1 spawned
$
   .
   .
   .
TIME FOR LUNCH
11-DEC-1995 12:00:56.99
```

The procedure prompts for a time value and for your message. Then the procedure spawns a subprocess that displays your message. You can continue working at your terminal; at the specified time, the subprocess rings the terminal bell, displays your message, and displays the time.

C.4 DIR.COM Command Procedure

C.4.1 Overview

This command procedure imitates the DCL command DIRECTORY/SIZE=ALL/DATE, displaying the block size (used and allocated) and creation date of the specified files. It illustrates use of the F$PARSE, F$SEARCH, F$FILE_ATTRIBUTES, and F$FAO lexical functions.

C.4.2 Example: DIR.COM

```
$ !
$ ! Command procedure implementation of DIRECTORY/SIZE=ALL/DATE
$ ! command
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ !
$ ! Replace any blank field of the P1 file specification with
$ ! a wildcard character
$ !
$ P1 = F$PARSE(P1,"*.*;*")                                    ❶
$ !
$ ! Define initial values for symbols
$ !
$ FIRST_TIME = "TRUE"
$ FILE_COUNT = 0
$ TOTAL_ALLOC = 0
$ TOTAL_USED = 0
$
$ LOOP:                                                       ❷
$       FILESPEC = F$SEARCH(P1)
$ ! Find next file in directory
$       IF FILESPEC .EQS. "" THEN GOTO DONE
$ ! If no more files, then done
$       IF .NOT. FIRST_TIME THEN GOTO SHOW_FILE
$ ! Print header only once
$ !
$ ! Construct and output the header line
$ !
$       FIRST_TIME =  "FALSE"                                 ❸
$       DIRSPEC = F$PARSE(FILESPEC,,, "DEVICE") -
                +F$PARSE(FILESPEC,,, "DIRECTORY")
$       WRITE SYS$OUTPUT ""
$       WRITE SYS$OUTPUT  "Directory ",DIRSPEC
$       WRITE SYS$OUTPUT ""
$       LASTDIR = DIRSPEC
$
$ !
$ ! Put the file name together, get some of the file attributes, and
$ ! type the information out
```

```
$ !
$SHOW_FILE:
$       FILE_COUNT = FILE_COUNT + 1
$       FILENAME = F$PARSE(FILESPEC,,, "NAME") -                    ❹
                 + F$PARSE(FILESPEC,,, "TYPE") -
                 + F$PARSE(FILESPEC,,, "VERSION")
$       ALLOC = F$FILE_ATTRIBUTES(FILESPEC, "ALQ")
$       USED = F$FILE_ATTRIBUTES(FILESPEC, "EOF")
$       TOTAL_ALLOC = TOTAL_ALLOC + ALLOC
$       TOTAL_USED = TOTAL_USED + USED
$       REVISED = F$FILE_ATTRIBUTES(FILESPEC,"RDT")
$       LINE = F$FAO("!19AS !5UL/!5<!UL!> !17AS",FILENAME,-
                USED, ALLOC, REVISED)
$       WRITE SYS$OUTPUT LINE
$       GOTO LOOP
$
$ !
$ ! Output summary information, reset verification, and exit
$ !
$ DONE:                                                             ❺
$       WRITE SYS$OUTPUT ""
$       WRITE SYS$OUTPUT "Total of ''FILE_COUNT' files, " + -
                         "''TOTAL_USED'/''TOTAL_ALLOC' blocks."
$       SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$       EXIT
```

C.4.3 Notes for DIR.COM Command Procedure

❶ This procedure uses the F$PARSE function to place asterisks in blank fields in P1, the user-supplied file specification. If you do not specify a parameter when you execute DIR.COM, then the F$PARSE function assigns the value ‘‘*.*;*’’ to P1. This causes DIR.COM to display all files in the current default directory.

❷ The F$SEARCH lexical function searches the directory for the file (or files) indicated by P1. If P1 contains any wildcards (asterisks), the F$SEARCH function returns all matching file specifications. After the last file specification has been returned, the procedure branches to the label DONE.

❸ The first time through the loop, the procedure writes a header for the directory display. This header includes the device and directory names. To obtain these names, the procedure uses the F$PARSE function.

❹ The procedure uses the F$PARSE lexical function to extract the file name from each file specification in the directory. The F$FILE_ATTRIBUTES lexical function then obtains blocks used, blocks allocated, and creation date information about each file. Finally, the F$FAO function formats a single display line for each file in the directory. The F$FAO function uses the file name and file attribute information as arguments.

❺ When F$SEARCH returns a null string, the procedure branches to the label DONE and displays summary information showing the total number of files, the total number of blocks used, and the total number of blocks allocated in the directory.

C.4.4 Sample Execution for DIR.COM Command Procedure

```
$ @DIR [VERN]*.COM

Directory DISK4:[VERN]

BATCH.COM;1              1/3     11-DEC-1995 11:43
CALC.COM;3               1/3     11-DEC-1995 11:30
CONVERT.COM;1            5/6     11-DEC-1995 15:23
   .
   .
   .
LOGIN.COM;34             2/3     11-DEC-1995 13:17
PID.COM;7                1/3     11-DEC-1995 09:49
SCRATCH.COM;6            1/3     11-DEC-1995 11:29)

Total of 15 files, 22/48 blocks.
```

The procedure returns information on all .COM files in the directory [VERN].

C.5 SYS.COM Command Procedure

C.5.1 Overview

This command procedure returns statistics about the current process, all processes in the group (if the current process has group privilege), and all processes on the system (if the current process has world privilege). This procedure illustrates use of the FPID, FEXTRACT, and F$GETJPI lexical functions.

C.5.2 Example: SYS.COM

```
$ !
$ ! Displays information about owner, group, or system processes.
$ !
$ ! Turn off verification and save current settings
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ CONTEXT = ""                    ! Initialize PID search context  ❶
$ !
$ ! Output header line.
$ !
$ WRITE SYS$OUTPUT "   PID    Username    Term    Process " + -     ❷
        "name   State   Pri Image"
$ !
$ ! Output process information.
$ !
$LOOP:
$ !
$ ! Get next PID.  If null, then done.
$ !
$       PID = F$PID(CONTEXT)                                      ❸
$       IF PID .EQS. "" THEN GOTO DONE
```

```
$ !
$ ! Get image file specification and extract the file name.
$ !
$       IMAGNAME = F$GETJPI(PID,"IMAGNAME")                    ❹
$       IMAGNAME = F$PARSE(IMAGNAME,,,"NAME","SYNTAX_ONLY")
$ !
$ ! Get terminal name.  If none, then describe type of process.
$ !
$       TERMINAL = F$GETJPI(PID,"TERMINAL")                    ❺
$       IF TERMINAL .EQS. "" THEN -
                TERMINAL = "-"+F$EXTRACT(0,3,F$GETJPI(PID,"MODE"))+"-"
$       IF TERMINAL .EQS. "-INT-" THEN TERMINAL = "-DET-"
$       IF F$GETJPI(PID,"OWNER") .NE. 0 THEN TERMINAL = "-SUB-"
$ !
$ ! Get more information, put process line together,
$ ! and output it.
$ !
$       LINE = F$FAO("!AS !12AS !7AS !15AS !5AS !2UL/!UL !10AS", - ❻
                PID,F$GETJPI(PID,"USERNAME"),TERMINAL,-
                F$GETJPI(PID,"PRCNAM"),-
                F$GETJPI(PID,"STATE"),F$GETJPI(PID,"PRI"),-
                F$GETJPI(PID,"PRIB"),IMAGNAME)
$       WRITE SYS$OUTPUT LINE
$       GOTO LOOP
$ !
$ ! Restore verification and exit.
$ !
$DONE:
$  SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$  EXIT
```

C.5.3 Notes for SYS.COM Command Procedure

❶ The symbol CONTEXT is initialized with a null value. This symbol will be used with the F$PID function to obtain a list of process identification numbers.

❷ The procedure writes a header for the display.

❸ The procedure gets the first process identification (PID) number. If the current process lacks group or world privilege, the PID number of the current process is returned. If the current process has group privilege, the first PID number in the group list is returned. The first PID number in the system list is returned if the current process has world privilege. The function continues to return the next PID number in sequence until the last PID number is returned. At this point, a null string is returned, and the procedure branches to the end.

❹ The procedure uses the F$GETJPI lexical function to get the image file specification for each PID number. The F$PARSE function extracts the file name from the specification returned by the F$GETJPI function.

❺ The procedure uses the F$GETJPI function to get the terminal name for each PID number. The F$EXTRACT function extracts the first three characters of the MODE specification returned by F$GETJPI(PID,"MODE") to determine the type of

process. The F$GETJPI function is used again to determine whether the process is a subprocess.

❻ The procedure uses the F$GETJPI lexical function to get the user name, process name, process state, process priority, and process base priority for each PID number returned. The F$FAO lexical function formats this information into a screen display.

C.5.4 Sample Execution for SYS.COM Command Procedure

```
$ @SYS

   PID     Username    Term    Process name   State   Pri Image
00050011  NETNONPRIV   -NET-    MAIL_14411      LEF    9/4 MAIL
00040013  STOVE        RTA6:    STOVE           LEF    9/4
00140015  MAROT        -DET-    DMFB0ACP        HIB    9/8 F11BAC
00080016  THOMPSON     -DET-    MTA0ACP         HIB   12/8 MTAAACP
00070017  JUHLES       TTF1:    JUHLES          LEF    9/4
   .
   .
   .
00040018  MARCO        TTA2:    MARCO           HIB    9/4 RTPAD
0018001A  VERN         RTA3:    VERN            LEF    9/4
0033001B  YISHA        RTA7:    YISHA           CUR    4/4
0002004A  SYSTEM       -DET-    ERRFMT          HIB   12/7 ERRFMT
```

This procedure returns information on all processes on the system. The current process has world privilege.

C.6 GETPARMS.COM Command Procedure

C.6.1 Overview

This command procedure returns the number of parameters passed to a procedure. You can call GETPARMS.COM from another procedure to determine how many parameters were passed to the calling procedure.

C.6.2 Example: GETPARMS.COM

```
$ ! Procedure to count the number of parameters passed to a command
$ ! procedure.  This number is returned as the global symbol PARMCOUNT.
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")          ❶
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ !
$ IF P1 .EQS. "?" THEN GOTO TELL                             ❷
$ !
$ ! Loop to count the number of parameters passed.  Null parameters are
$ ! counted until the last non-null parameter is passed.
$ !
```

```
$       COUNT = 0                                                    ❸
$       LASTNONNULL = 0
$ LOOP:
$       IF COUNT .EQ. 8 THEN GOTO END_COUNT
$       COUNT = COUNT + 1
$       IF P'COUNT' .NES. "" THEN LASTNONNULL = COUNT
$ GOTO LOOP
$ !
$ END_COUNT:                                                         ❹
$ !
$ ! Place the number of non-null parameters passed into PARMCOUNT.
$ !
$ PARMCOUNT == LASTNONNULL
$ !
$ ! Restore verification setting, if it was on, before exiting
$ !                                                                  ❺
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$ EXIT
$ !
$ TELL:                                                              ❻
$ TYPE SYS$INPUT
        This procedure counts the number of parameters passed to
        another procedure.  This procedure can be called by entering
        the following string in any procedure:

             @GETPARMS 'P1 'P2 'P3 'P4 'P5 'P6 'P7 'P8

        On  return,  the  global  symbol  PARMCOUNT
        contains the number of parameters passed to the procedure.
$ !
$ EXIT
```

C.6.3 Notes for GETPARMS.COM Command Procedure

❶ The procedure saves the current image and procedure verification settings before turning off verification.

❷ If a question mark character was passed to the procedure as a parameter, the procedure branches to the label TELL (Note 6).

❸ A loop is established to count the number of parameters that were passed to the procedure. The counters COUNT and LASTNONNULL are initialized to 0 before entering the loop. Within the loop, COUNT is incremented and tested against the value 8. If COUNT is equal to 8, the maximum number of parameters has been entered. Each time a non-null parameter is passed, LASTNONNULL is equated to that parameter's number.

Each time the IF command executes, the symbol COUNT has a different value. The first time, the value of COUNT is 1 and the IF command checks P1. The second time, it checks P2, and so on.

❹ When the parameter count reaches 8, the procedure branches to END_COUNT. The symbol LASTNONNULL contains the count of the last non-null parameter passed. This value is placed in the global symbol PARMCOUNT. PARMCOUNT

must be defined as a global symbol so that its value can be tested at the calling command level.

❺ The original verification settings are restored.

❻ At the label TELL, the TYPE command displays data that is included in the input stream. (In command procedures, the input stream is the command procedure file.) The TYPE command displays instructions on how to use GETPARMS.COM.

C.6.4 Sample Execution for GETPARMS.COM Command Procedure

The procedure SORTFILES.COM requires the user to pass three non-null parameters. The SORTFILES.COM procedure can contain the following lines:

```
$ @GETPARMS 'P1' 'P2' 'P3' 'P4' 'P5' 'P6' 'P7' 'P8'
$ IF PARMCOUNT .NE. 3 THEN GOTO NOT_ENOUGH
   .
   .
   .
$NOT_ENOUGH:
$ WRITE SYS$OUTPUT -
"Three non-null parameters required.  Type SORTFILES HELP for info."
$ EXIT
```

The procedure SORTFILES.COM can be invoked as follows:

```
$ @SORTFILES DEF 4
Three non-null parameters required. Type SORTFILE HELP for info.
```

For this procedure to be properly invoked—that is, for the parameters that are passed to SORTFILES to be passed intact to GETPARMS for processing—the symbols P1 to P8 must be enclosed in single quotation marks.

If the return value from GETPARMS is not 3, SORTFILES outputs an error message and exits.

C.7 EDITALL.COM Command Procedure

C.7.1 Overview

Invokes the EDT editor repeatedly to edit a group of files with the same file type. This procedure illustrates how to use lexical functions to extract file names from columnar output. It also illustrates a way to redefine the input stream for a program invoked within a command procedure.

C.7.2 Example: EDITALL.COM

```
$ ! Procedure to edit all files in a directory with a
$ ! specified file type. Use P1 to indicate the file type.
$ !
$ ON CONTROL_Y THEN GOTO DONE           ! Ctrl/Y action    ❶
$ ON ERROR THEN GOTO DONE
```

```
$ !
$ ! Check for file type parameter.  If one was entered, continue;
$ ! otherwise, prompt for a parameter.
$ !
$ IF P1 .NES. "" THEN GOTO OKAY                                    ❷
$ INQUIRE P1 "Enter file type of files to edit"
$ !
$ ! List all files with the specified file type and write the DIRECTORY
$ ! output to a file named DIRECT.OUT
$ !
$ OKAY:
$ DIRECTORY/VERSIONS=1/COLUMNS=1 -                                 ❸
        /NODATE/NOSIZE -
        /NOHEADING/NOTRAILING -
        /OUTPUT=DIRECT.OUT *.'P1'
$ IF .NOT. $STATUS THEN GOTO ERROR_SEC                             ❹
$ !
$ OPEN/READ/ERROR=ERROR_SEC DIRFILE DIRECT.OUT                     ❺
$ !
$ ! Loop to read directory file
$ !
$ NEWLINE:                                                         ❻
$       READ/END=DONE DIRFILE NAME
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:   ! Redefine SYS$INPUT
$       EDIT 'NAME'                               ! Edit the file
$       GOTO NEWLINE
$ !
$ DONE:                                                            ❼
$       CLOSE DIRFILE/ERROR=NOTOPEN               ! Close the file
$ NOTOPEN:
$       DELETE DIRECT.OUT;*                       ! Delete temp file
$ EXIT
$ !
$ ERROR_SEC:
$       WRITE SYS$OUTPUT "Error:  ",F$MESSAGE($STATUS)
$       DELETE DIRECT.OUT;*
$ EXIT
```

C.7.3 Notes for EDITALL.COM Command Procedure

❶ ON commands establish condition handling for this procedure. If Ctrl/Y is pressed at any time during the execution of this procedure, the procedure branches to the label DONE. Similarly, if any error or severe error occurs, the procedure branches to the label DONE.

❷ The procedure checks whether a parameter was entered. If not, the procedure prompts for a file type.

❸ The DIRECTORY command lists all files with the file type specified as P1. The command output is written to the file DIRECT.OUT. The /VERSIONS=1 qualifier requests that only the highest numbered version of each file be listed. The /NOHEADING and /NOTRAILING qualifiers request that no heading lines or directory summaries be included in the output. The /COLUMNS=1 qualifier ensures that one file name per record is given.

❹ The IF command checks the return value from the DIRECTORY command by testing the value of $STATUS. If the DIRECTORY command does not complete successfully, then $STATUS has an even integer value, and the procedure branches to the label ERROR_SEC.

❺ The OPEN command opens the directory output file and gives it a logical name of DIRFILE.

❻ The READ command reads a line from the DIRECTORY command output into the symbol name NAME. After it reads each line, the procedure uses the DEFINE command to redefine the input stream for the edit session (SYS$INPUT) to be the terminal. Then, it invokes the editor, specifying the symbol NAME as the file specification. When the edit session is completed, the command interpreter reads the next line in the command procedure and branches to the label NEWLINE. When the procedure has edited all files of the specified file type in the directory, it branches to the label DONE.

❼ The label DONE is the target label for the /END qualifier on the READ command and the target label for the ON CONTROL_Y and ON ERROR conditions set at the beginning of the procedure. At this label, the procedure performs the necessary cleanup operations.

The CLOSE command closes the DIRECTORY command output file; the /ERROR qualifier specifies the label on the next line in the file. This use of /ERROR suppresses any error message that would be displayed if the directory file is not open. For example, this would occur if Ctrl/Y were pressed before the directory file were opened.

The second step in cleanup is to delete the temporary directory file.

C.7.4 Sample Execution for EDITALL.COM Command Procedure

```
$ @EDITALL DAT
*   .
    .
    .
%DELETE-I-FILDEL, device:[directory]DIRECT.OUT;1 deleted (x blocks)
```

The procedure EDITALL is invoked with P1 specified as .DAT. The procedure creates a directory listing of all files in the default directory whose file types are .DAT and invokes the editor to edit each one. After you finish editing the last file with the file type .DAT, the procedure deletes the temporary file DIRECT.OUT and displays an informational message at your terminal.

C.8 MAILEDIT.COM Command Procedure

C.8.1 Overview

This command procedure invokes a text editor in the Mail utility.

C.8.2 Example: MAILEDIT.COM

```
$ ! Command procedure to invoke an editor for Mail.
$ !
$ ! Inputs:
$ !
$ ! P1 = Input file name.
$ ! P2 = Output file name.
$ !
$ ! If MAIL$EDIT is undefined, Mail will invoke the user's selected
$ ! callable editor set by the mail SET EDITOR command.
$ !
$ ! If MAIL$EDIT is defined to be a command procedure, Mail will create
$ ! a subprocess to edit the mail, but any SET EDITOR command in Mail
$ ! will override the definition of MAIL$EDIT for the remainder of that
$ ! Mail session.
$ !
$ ! Note that this procedure is run in the context of a subprocess.
$ ! LOGIN.COM is not executed.  However, all process logical names
$ ! and DCL global symbols are copied.  In particular, note that the
$ ! user's individual definition of the symbol EDIT is used if there
$ ! is one. Otherwise, the system default editor is used.
$ !
$ ! The default directory is the same as the parent process
$ !
$ DEFINE /USER SYS$INPUT 'F$TRNLNM("SYS$OUTPUT")'     ❶
$ IF P1 .EQS. "" THEN GOTO NOINPUT       ❷
$ EDIT /OUTPUT='P2' 'P1'       ❸
$ EXIT
$NOINPUT:
$ EDIT 'P2'         ❹
$ EXIT
```

C.8.3 Notes for MAILEDIT.COM Command Procedure

❶ The DEFINE command allows the editor input to come from the terminal instead of the command file.

❷ The IF statement distinguishes between editing a file with a different output file name from editing a file with the same file name.

❸ This EDIT command invokes an editor with input and output file names. You can edit this line to invoke an editor of your choice. For example:

```
$ RUN XYZ_EDITOR.EXE /INPUT= 'P1' /OUTPUT='P2'
```

❹ This EDIT command invokes an editor with a single file name. You can edit this line to invoke an editor of your choice. For example:

```
$ RUN XYZ_EDITOR.EXE /INPUT= 'P2' /OUTPUT='P2'
```

C.8.4 Example: Execution for MAILEDIT.COM Command Procedure

```
$DEFINE MAIL$EDIT MAILEDIT.COM
$MAIL
MAIL> SHOW EDITOR
Your editor is defined by the file MAILEDIT.COM.
```

C.9 FORTUSER.COM Command Procedure

C.9.1 Overview

Provides a sample of a system-defined login command procedure that controls the terminal environment for an interactive user who creates, compiles, and executes FORTRAN programs. If a user logs into a captive account where FORTUSER.COM is listed as the login command procedure, the user can execute only the commands accepted by FORTUSER.COM. This procedure also illustrates how to use lexical functions to step through an option table, comparing a user-entered command with a list of valid commands.

C.9.2 Example: FORTUSER.COM

```
$ ! Procedure to create, compile, link, execute, and debug
$ ! FORTRAN programs.  Users can enter only the commands listed
$ ! in the symbol OPTION_TABLE.
$ SET NOCONTROL=Y                                                    ❶
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ OPTION_TABLE = "EDIT/COMPILE/LINK/RUN/EXECUTE/DEBUG/PRINT/HELP/FILE/DONE/"  ❷
$ TYPE SYS$INPUT                                                     ❸

       VMS FORTRAN Command Interpreter

       Enter name of file with which you would like to work.
$ !
$ ! Set up for initial prompt
$ !
$ PROMPT = "INIT"                                                    ❹
$ GOTO HELP                     ! Print the initial help message
$ !
$ ! after the first prompting message, use the prompt: Command
$ !
$ INIT:
$ PROMPT = "GET_COMMAND"
$ GOTO FILE                     ! Get initial file name
$ !
$ ! Main command parsing routine.  The routine compares the current
$ ! command against the options in the option table.  When it finds
$ ! a match, it branches to the appropriate label.
$ !
$ GET_COMMAND:
$       ON CONTROL_Y THEN GOTO GET_COMMAND     ! Ctrl/Y resets prompt  ❺
$       SET CONTROL=Y
$       ON WARNING THEN GOTO GET_COMMAND       ! If any, reset prompt
$       INQUIRE COMMAND "Command"
$       IF COMMAND .EQS. "" THEN GOTO GET_COMMAND
$       IF F$LOCATE(COMMAND + "/", OPTION_TABLE) .EQ. F$LENGTH(OPTION_TABLE) - ❻
              THEN GOTO INVALID_COMMAND
$       GOTO 'COMMAND'
$ !
$ INVALID_COMMAND:                                                   ❼
$       WRITE SYS$OUTPUT " Invalid command"
$ !
```

```
$ HELP:                                                          ❽
$       TYPE SYS$INPUT
        The commands you can enter are:
        FILE      Name of FORTRAN program in your current
                  default directory.  Subsequent commands
                  process this file.
        EDIT      Edit the program.
        COMPILE   Compile the program with FORTRAN.
        LINK      Link the program to produce an executable image.
        RUN       Run the program's executable image.
        EXECUTE   Same function as COMPILE, LINK, and RUN.
        DEBUG     Run the program under control of the debugger.
        PRINT     Queue the most recent listing file for printing.
        DONE      Return to interactive command level.
        HELP      Print this help message.

        Enter Ctrl/Y to restart this session
$ GOTO 'PROMPT'                                                  ❾
$ EDIT:                                                          ❿
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       EDIT 'FILE_NAME'.FOR
$       GOTO GET_COMMAND
$ COMPILE:
$       FORTRAN 'FILE_NAME'/LIST/OBJECT/DEBUG
$       GOTO GET_COMMAND
$ LINK:
$       LINK 'FILE_NAME'/DEBUG
$       PURGE 'FILE_NAME'.*/KEEP=2
$       GOTO GET_COMMAND
$ RUN:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN/NODEBUG 'FILE_NAME'
$       GOTO GET_COMMAND
$ DEBUG:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN 'FILE_NAME'
$       GOTO GET_COMMAND
$ EXECUTE:
$       FORTRAN 'FILE_NAME'/LIST/OBJECT
$       LINK/DEBUG 'FILE_NAME'
$       PURGE 'FILE_NAME'.*/KEEP=2
$       RUN/NODEBUG 'FILE_NAME'
$       GOTO GET_COMMAND
$ PRINT:
$       PRINT 'FILE_NAME'
$       GOTO GET_COMMAND
$ BADFILE:                                                       ⓫
$       WRITE SYS$OUTPUT "File must be in current default directory."
$ FILE:
$       INQUIRE FILE_NAME "File name"
$       IF FILE_NAME .EQS. "" THEN GOTO FILE
$       IF F$PARSE(FILE_NAME,,,"DIRECTORY") .NES. F$DIRECTORY() -  ⓬
        THEN GOTO BADFILE
$       FILE_NAME = F$PARSE(FILE_NAME,,,"NAME")
$       GOTO GET_COMMAND
$ DONE:
$ EXIT
```

C.9.3 Notes for FORTUSER.COM Command Procedure

❶ The SET NOCONTROL=Y command ensures that the user who logs in under the control of this procedure cannot interrupt the procedure or any command or program in it.

❷ The option table lists the commands that the user will be allowed to execute. Each command is separated by a slash.

❸ The procedure introduces itself.

❹ The symbol name PROMPT is given the value of a label in the procedure. When the procedure is initially invoked, this symbol has the value INIT. The HELP command text terminates with a GOTO command that specifies the label PROMPT. When this text is displayed for the first time, the GOTO command results in a branch to the label HELP. This displays the HELP message that explains the commands that you can enter. Then, the procedure branches back to the label INIT, where the value for PROMPT is changed to "GET_COMMAND". Finally, the procedure branches to the label FILE to get a file name. Thereafter, when the help text is displayed, the procedure branches to the label GET_ COMMAND to get the next command.

❺ The Ctrl/Y condition action is set to return to the label GET_COMMAND, as is the warning condition action. The procedure prompts for a command and continues to prompt, even if nothing is entered. To terminate the session and return to interactive command level, enter the command DONE.

❻ The procedure uses the F$LOCATE and F$LENGTH lexical functions to determine whether command is included in the list of options. The F$LOCATE function searches for the user-entered command, followed by a slash. (For example, if you enter EDIT, the procedure searches for EDIT/.) If the command is not included in the option list, then the procedure branches to the label INVALID_COMMAND. If the command is valid, the procedure branches to the appropriate label.

❼ At the label INVALID_COMMAND, the procedure writes an error message and displays the help text that lists the commands that are valid.

❽ The help text lists the commands that are valid. This text is displayed initially. It is also displayed whenever the user enters the HELP command or any invalid command.

❾ At the conclusion of the HELP text, the GOTO command specifies the symbol name PROMPT. When this procedure is first invoked, the symbol has the value INIT. Thereafter, it has the value GET_COMMAND.

❿ Each valid command in the list has a corresponding entry in the option table and a corresponding label in the command procedure. For the commands that read input from the terminal, for example, EDIT, the procedure contains

a DEFINE command that defines the input stream as SYS$COMMAND.

⓫ At the label BADFILE, the procedure displays a message indicating that the file must be in the current directory. Then the procedure prompts for another file name.

⓬ After obtaining a file name, the procedure checks that you have not specified a directory that is different from your current default directory. The procedure then uses the F$PARSE function to extract the file name. (Each command supplies the appropriate default file type.) Next, the procedure branches back to the label GET_COMMAND to get a command to process the file.

C.9.4 Sample Execution for FORTUSER.COM Command Procedure

The following example illustrates how to use this command procedure as a captive command procedure:

```
Username: CLASS30
Password:

            OpenVMS Version 6.2

    OpenVMS FORTRAN Command Interpreter

    Enter name of file with which you would like to work.

    The commands you can enter are:

    FILE       Name of FORTRAN program in your current
               default directory.  Subsequent commands
               process this file.
    EDIT       Edit the program.
    COMPILE    Compile the program with VAX FORTRAN.
    LINK       Link the program to produce an executable image.
    RUN        Run the program's executable image.
    EXECUTE    Same function as COMPILE, LINK and RUN.
    DEBUG      Run the program under control of the debugger.
    PRINT      Queue the most recent listing file for printing.
    DONE       Return to interactive command level.
    HELP       Print this help message.

    Enter Ctrl/Y to restart this session
File name: AVERAGE
Command: COMPILE
Command: LINK
Command: RUN
Command: FILE
File name: READFILE
Command: EDIT
```

This sample execution illustrates a session in which a user named CLASS30 logs in to the account controlled by the FORTUSER command procedure. The FORTUSER command procedure displays the commands the user is allowed to execute, as well as an instruction for restarting the session. Next, the user specifies the file AVERAGE, compiles, links, and runs it. Then, the user enters the FILE command to begin working on another file.

C.10 LISTER.COM Command Procedure

C.10.1 Overview

Prompts for input data, formats the data in columns, and sorts it into an output file. This procedure illustrates the READ and WRITE commands, as well as the character substring overlay format of an assignment statement.

C.10.2 Example: LISTER.COM

```
$ ! Procedure to accumulate programmer names and document file names.
$ ! After all programmer names and file names are entered, they are
$ ! sorted in alphabetic order by programmer name and printed on
$ ! the system printer.
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")          (1)
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ !
$ OPEN/WRITE OUTFILE DATA.TMP         ! Create output file  (2)
$ !
$ ! Loop to obtain programmers' last names and file names,
$ ! and write this data to DATA.TMP.
$ !
$ LOOP:                                                    (3)
$        INQUIRE NAME "Programmer (press Return to quit)"
$        IF NAME .EQS. "" THEN GOTO FINISHED
$        INQUIRE FILE "Document file name"
$        RECORD[0,20]:='NAME'                              (4)
$        RECORD[21,20]:='FILE'
$        WRITE OUTFILE RECORD
$        GOTO LOOP
$ FINISHED:
$        CLOSE OUTFILE
$ !
$ DEFINE/USER_MODE SYS$OUTPUT: NL:   ! Suppress sort output
$ SORT/KEY=(POSITION:1,SIZE=20) DATA.TMP DOC.SRT           (5)
$ !
$ OPEN/WRITE OUTFILE DOCUMENT.DAT                          (6)
$ WRITE OUTFILE "Programmer Files as of ",F$TIME()
$ WRITE OUTFILE ""
$ RECORD[0,20]:="Programmer Name"
$ RECORD[21,20]:="File Name"
$ WRITE OUTFILE RECORD
$ WRITE OUTFILE ""
$ !
$ CLOSE OUTFILE                                            (7)
$ APPEND DOC.SRT DOCUMENT.DAT
$ PRINT DOCUMENT.DAT
$ !
$ INQUIRE CLEAN_UP "Delete temporary files [Y,N]"          (8)
$ IF CLEAN_UP THEN DELETE DATA.TMP;*,DOC.SRT;*
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$ EXIT
```

C.10.3 Notes for LISTER.COM Command Procedure

❶ LISTER.COM saves the current verification setting and turns off verification.

❷ The OPEN command creates a temporary file for writing.

❸ INQUIRE commands prompt for a programmer name and for a file name. If a null line, signaled by pressing Return, is entered in response to the INQUIRE command prompt, the procedure assumes that no more data is to be entered and branches to the label FINISHED.

❹ After assigning values to the symbols NAME and FILE, the procedure uses the character string overlay format of an assignment statement to construct a value for the symbol RECORD. In columns 1 to 21 of RECORD, the current value of NAME is written. The command interpreter pads the value of NAME with spaces to fill the 20-character length specified.

Similarly, the next 20 columns of RECORD are filled with the value of FILE. Then, the value of RECORD is written to the output file.

❺ After the file has been closed, the procedure sorts the output file DATA.TMP. The DEFINE command directs the SORT command output to the null file NL. Otherwise, these statistics would be displayed on the terminal:

The sort is performed on the first 20 columns; that is, by programmer name.

The sorted output file has the name DOC.SRT.

❻ The procedure creates the final output file, DOCUMENT.DAT, with the OPEN command. The first lines written to the file are header lines, giving a title, the date and time of day, and headings for the columns.

❼ The procedure closes the file DOCUMENT.DAT and appends the sorted output file, DOC.SRT, to it. Then, the output file is queued to the system printer.

❽ The procedure prompts to determine whether to delete the intermediate files. If a true response (T, t, Y, or y) is entered at the INQUIRE command prompt, the files DATA.TMP and DOC.SRT are deleted. Otherwise, they are retained.

C.10.4 Sample Execution for LISTER.COM Command Procedure

```
$ @LISTER
Programmer: WATERS
Document file name: CRYSTAL.CAV
Programmer: JENKINS
Document file name: MARIGOLD.DAT
Programmer: MASON
Document file name: SYSTEM.SRC
Programmer: ANDERSON
Document file name: JUNK.J
Programmer: Return
Delete temporary files [Y,N]:y
```

The output file resulting from this execution of the procedure contains the following:

```
Programmer Files as of 31-DEC-1995 16:18:58.79

Programmer Name      File Name

ANDERSON             JUNK.J
JENKINS              MARIGOLD.DAT
MASON                SYSTEM.SRC
WATERS               CRYSTAL.CAV
```

C.11 CALC.COM Command Procedure

C.11.1 Overview

Performs arithmetic calculations and converts the resulting value to hexadecimal and decimal values.

C.11.2 Example: CALC.COM

```
$ ! Procedure to calculate expressions.  If you enter an
$ ! assignment statement, then CALC.COM evaluates the expression
$ ! and assigns the result to the symbol you specify.  In the next
$ ! iteration, you can use either your symbol or the symbol Q to
$ ! represent the current result.
$ !
$ ! If you enter an expression, then CALC.COM evaluates the
$ ! expression and assigns the result to the symbol Q.  In
$ ! the next iteration, you can use the symbol Q to represent
$ ! the current result.
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ START:
$     ON WARNING THEN GOTO START                                    ❶
$     INQUIRE STRING "Calc"                                         ❷
$     IF STRING .EQS. "" THEN GOTO CLEAN_UP
$     IF F$LOCATE("=",STRING) .EQ. F$LENGTH(STRING) THEN GOTO EXPRESSION
```

```
$ !
$ ! Execute if string is in the form symbol = expression
$ STATEMENT:                                                    ❸
$     'STRING' ! Execute assignment statements
$     SYMBOL = F$EXTRACT(0,F$LOCATE("=",STRING)-1,STRING) ! get symbol name
$     Q = 'SYMBOL'        ! Set up q for future iterations
$     LINE = F$FAO("Decimal = !SL       Hex = !-!XL      Octal = !-!OL",Q)
$     WRITE SYS$OUTPUT LINE
$     GOTO START
$ !
$ !
$ ! Execute if string is an expression
$ EXPRESSION:                                                   ❹
$     Q = F$INTEGER('STRING')         ! Can use Q in next iteration
$     LINE = F$FAO("Decimal = !SL       Hex = !-!XL      Octal = !-!OL",Q)
$     WRITE SYS$OUTPUT LINE
$     GOTO START
$ !
$ CLEAN_UP:
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$ EXIT
```

C.11.3 Notes for CALC.COM Command Procedure

❶ The procedure establishes an error-handling condition that restarts the procedure. If a warning or an error of greater severity occurs, the procedure will branch to the beginning where it resets the ON condition.

This technique ensures that the procedure will not exit if the user enters an expression incorrectly.

❷ The INQUIRE command prompts for an arithmetic expression. The procedure accepts expressions in either of the following formats:

name = expression

expression

If you press Return, the procedure assumes the end of a CALC session and exits.

If you enter input in the format "name = expression" then the procedure continues executing at the label STATEMENT. Otherwise, the procedure branches to the label EXPRESSION.

❸ The procedure executes the assignment statement and assigns the result of the expression to the symbol. Then the procedure extracts the symbol name, and assigns the value of the symbol to Q. This allows you to use either Q or your symbol during the next iteration of the procedure. Next, the procedure displays the result and then branches back to the label START.

❹ The procedure evaluates the expression and assigns the result to the symbol Q. This allows you to use Q during the next iteration of the procedure. Next, the procedure displays the result and then branches back to the label START.

C.11.4 Sample Execution for CALC.COM Command Procedure

```
$ @CALC
Calc: 2 * 30
Decimal = 60        Hex = 0000003C  Octal = 00000000074
Calc: Q + 3
Decimal = 63        Hex = 0000003F  Octal = 00000000077
Calc: TOTAL = Q + 4
Decimal = 67        Hex = 00000043  Octal = 00000000103
Calc: 5 + 7
Decimal = 12    Hex = 0000000C  Octal = 00000000014
Calc: Return
$
```

After each prompt from the procedure, the user enters an arithmetic expression. The procedure displays the results in decimal, hexadecimal, and octal. A null line, signaled by pressing Return on a line with no data, concludes the CALC session.

C.12 BATCH.COM Command Procedure

C.12.1 Overview

Accepts a command string, a command procedure, or a list of commands and then executes these commands as a batch job.

C.12.2 Example: BATCH.COM

```
$ VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ VERIFY_PROCEDURE = F$VERIFY(0)
$!
$! Turn off verification and save current settings.
$! (This comment must appear after you turn verification
$! off; otherwise it will appear in the batch job log file.)
$!
$!
$! If this is being executed as a batch job,
$! (from the SUBMIT section below) go to the EXECUTE_BATCH_JOB section
$! Otherwise, get the information you need to prepare to execute the
$! batch job.
$!
$ IF F$MODE() .EQS. "BATCH" THEN GOTO EXECUTE_BATCH_JOB    ❶
$!
$!
$! Prepare to submit a command (or a command procedure) as a batch job.
$! First, determine a mnemonic process name for the batch job.  Use the
$! following rules:
$!
$! 1) If the user is executing a single command, then use the verb name.
$!    Strip off any qualifiers that were included with the command.
$! 2) If the user is executing a command procedure, then use the file name.
$! 3) Otherwise, use BATCH.
$!
$ JOB_NAME = P1         ❷
$ IF JOB_NAME .EQS. "" THEN JOB_NAME = "BATCH"
$ IF F$EXTRACT(0,1,JOB_NAME) .EQS. "@" THEN JOB_NAME = F$EXTRACT(1,999,JOB_NAME)
$ JOB_NAME = F$EXTRACT(0,F$LOCATE("/",JOB_NAME),JOB_NAME)
$ JOB_NAME = F$PARSE(JOB_NAME,,,"NAME","SYNTAX_ONLY")
$ IF JOB_NAME .EQS. "" THEN JOB_NAME = "BATCH"
$!
```

```
$!
$! Get the current default device and directory.
$!
$ ORIGDIR = F$ENVIRONMENT("DEFAULT")
$!
$!
$! Concatenate the parameters to form the command string to be executed.
$! If the user did not enter a command string, the symbol COMMAND will have
$! a null value.
$!
$ COMMAND = P1 + " " + P2 + " " + P3 + " " + P4 + " " + -  ❸
            P5 + " " + P6 + " " + P7 + " " + P8
$!
$!
$! If the user is executing a single command and if both the command and the
$! original directory specification are small enough to be passed as
$! parameters to the SUBMIT command, then submit the batch job now
$!
$ IF (P1 .NES. "") .AND. (F$LENGTH(COMMAND) .LE. 255) .AND. -    ❹
        (F$LENGTH(ORIGDIR) .LE. 255) THEN GOTO SUBMIT
$!
$!
$! If the single command to be executed in the batch job is very large, or
$! if you have to prompt for commands to execute in the batch job, then
$! create a temporary command procedure to hold those commands and get the
$! fully expanded name of the command procedure.
$!
$ CREATE_TEMP_FILE:
$   ON CONTROL_Y THEN GOTO CONTROL_Y_HANDLER                ❺
$   OPEN/WRITE/ERROR=FILE_OPEN_ERROR TEMPFILE SYS$SCRATCH:'JOB_NAME'.TMP ❻
$   FILESPEC = F$SEARCH("SYS$SCRATCH:" + JOB_NAME + ".TMP")
$!
$!  By default, have the batch job continue if it encounters any errors.
$!
$   WRITE TEMPFILE "$ SET NOON"
$!
$!  Either write the single large command to the file, or prompt for
$!  multiple commands and write them to the file.
$!
$   IF COMMAND .NES. "       " THEN GOTO WRITE_LARGE_COMMAND
$
$   LOOP:
$       READ /END_OF_FILE=CLOSE_FILE /PROMPT="Command: " SYS$COMMAND COMMAND
$       IF COMMAND .EQS. "" THEN GOTO CLOSE_FILE
$       WRITE TEMPFILE "$ ",COMMAND
$       GOTO LOOP
$
$ WRITE_LARGE_COMMAND:
$   WRITE TEMPFILE "$ ",COMMAND
$
$!
$! Finish the temporary file by defining a symbol so that you will know
$! the name of the command procedure to delete and then close the file.
$! Define the symbol COMMAND to mean "execute the command procedure
$! you have just created".  Then submit the batch job and execute
$! this command procedure in the batch job.
$!
```

```
$ CLOSE_FILE:                                                    ❼
$   WRITE TEMPFILE "$ BATCH$DELETE_FILESPEC == """,FILESPEC,""""
$   CLOSE TEMPFILE
$   ON CONTROL_Y THEN EXIT
$   COMMAND = "@" + FILESPEC
$!
$!
$! Submit BATCH.COM as a batch job, and pass it two parameters.
$! P1 is the command (or name of the command procedure) to execute.
$! P2 is the directory from which to execute the command.
$!
$ SUBMIT:                                                        ❽
$   SUBMIT/NOTIFY/NOPRINT 'F$ENVIRONMENT("PROCEDURE")' /NAME='JOB_NAME' -
        /PARAMETERS=("''COMMAND'","''ORIGDIR'")
$   GOTO EXIT
$!
$!
$! The user pressed Ctrl/Y while the temporary command procedure was open.
$! Close the command procedure, delete it if it exists, and exit.
$!
$ CONTROL_Y_HANDLER:                                             ❾
$   CLOSE TEMPFILE
$   IF F$TYPE(FILESPEC) .NES. "" THEN DELETE/NOLOG 'FILESPEC'
$   WRITE SYS$OUTPUT "Ctrl/Y caused the command procedure to abort."
$   GOTO EXIT
$!
$!
$! The temporary command procedure could not be created.
$! Notify the user and exit.
$!
$ FILE_OPEN_ERROR:                                               ❿
$   WRITE SYS$OUTPUT "Could not create sys$scratch:",job_name,".tmp"
$   WRITE SYS$OUTPUT "Please correct the situation and try again."
$!
$!
$! Restore the verification states and exit.
$!
$ EXIT:                                                          ⓫
$  VERIFY_PROCEDURE = F$VERIFY(VERIFY_PROCEDURE,VERIFY_IMAGE)
$  EXIT
$!
$!
$! BATCH.COM was invoked as a batch job.  P1 contains the command
$! to execute and P2 the default directory specification.
$! Return a status code that indicates the termination status of P1.
$!
$ EXECUTE_BATCH_JOB:                                             ⓬
$    SET NOON
$    VERIFY_PROCEDURE = F$VERIFY(VERIFY_PROCEDURE,VERIFY_IMAGE)
$    SET DEFAULT 'P2'
$    'P1'
$    IF F$TYPE(BATCH$DELETE_FILESPEC) .EQS. "" THEN EXIT $STATUS
$    STATUS = $STATUS
$    DELETE /NOLOG 'BATCH$DELETE_FILESPEC'
$    EXIT STATUS
```

C.12.3 Notes for BATCH.COM Command Procedure

❶ This IF statement tests whether BATCH.COM is executing in batch mode. When you invoke BATCH.COM interactively, you provide (as parameters) a command string or a command procedure that is to be executed as a batch job. If you do not supply any parameters, then BATCH.COM prompts you for commands, writes these commands to a file, and then executes this command procedure as a batch job. After BATCH.COM prepares your commands for execution from a batch job, it uses the SUBMIT command to submit itself as a batch job and execute your commands from this job. (See Note 8.) When you invoke BATCH.COM as a batch job, the procedure branches to the label EXECUTE_BATCH_JOB. Note that you must specify the command or command procedure to execute as P1 and the default directory as P2 if you execute BATCH.COM in batch mode.

❷ These commands prepare the batch job for execution. First, the procedure constructs a name for the batch job. If a command string was passed, then BATCH.COM uses the verb name as the job name. If a command procedure was passed, then BATCH.COM uses the file name. If no input was passed, then BATCH.COM names the job BATCH.

❸ The parameters are concatenated to form the command string to be executed. The command string is assigned to the symbol COMMAND.

❹ The SUBMIT command cannot pass a parameter that is greater than 255 characters. Therefore, the procedure tests that the command string and directory name are less than 255 characters long. If both strings are less than 255 characters (and if the user did, in fact, pass a command string), then the procedure branches to the label SUBMIT.

❺ The procedure establishes a Ctrl/Y handler, so cleanup operations are performed if the user presses Ctrl/Y during this section of the command procedure.

❻ The procedure creates a temporary file to contain the commands to be executed. If the user supplies a long command string, the procedure branches to WRITE_LARGE_ COMMAND and writes this command to the temporary file. Otherwise, the procedure prompts for the commands to be executed. Each command is written to the temporary file.

❼ Before you close the temporary file, write a symbol assignment statement to the file. This statement assigns the file name for the temporary file to the symbol BATCH$DELETE_ FILESPEC. After closing the temporary file, the procedure resets the default Ctrl/Y handler. Then the procedure defines the symbol COMMAND so that, when executed, the symbol COMMAND invokes the temporary command file.

❽ The procedure submits itself as a batch job, using the defined job name. (See Note 2.) The procedure also passes two parameters: the command or command procedure to be executed, and the directory from which the command should be executed. Then the procedure branches to the label EXIT. (See Note 11.)

❾ This section performs clean-up operations if the user enters Ctrl/Y while the temporary file is being created.

❿ This section writes an error message if the temporary file cannot be created.

⓫ The procedure resets the original verification settings and then exits.

⓬ These commands are executed when BATCH.COM runs in batch mode. First, ON error handling is disabled and the user's default verification settings are set. Then the default is set to the directory indicated by P2, and the command or command procedure indicated by P1 is executed. If a temporary file was created, this file is deleted. The completion status for P1 is saved before deleting BATCH$DELETE_FILESPEC. This completion status is returned by the EXIT command.

C.12.4 Sample Execution for BATCH.COM Command Procedure

```
$ @BATCH RUN MYPROG
Job RUN (queue SYS$BATCH, entry 1715) started on SYS$BATCH
```

The following example uses BATCH.COM to run a program from within a batch job.

C.13 COMPILE_FILE.COM Command Procedure

C.13.1 Overview

Compiles, links, and runs a file written in Pascal or FORTRAN. It prompts for a file to process and determines if the file type is .PAS or .FOR. If the file type is not .PAS or .FOR, or if the file does not exist in the current default directory, the command procedure outputs appropriate error messages. This procedure illustrates the use of the IF-THEN-ELSE language construct.

C.13.2 Example: COMPILE_FILE.COM

```
$! This command procedure compiles, links, and runs a file written in Pascal
$! or FORTRAN.
$!
$ ON CONTROL_Y THEN EXIT
$!
$ TOP:
$   INQUIRE FILE "File to process"
$   IF F$SEARCH(FILE) .NES. ""                                    ❶
$ THEN
$   FILE_TYPE = F$PARSE(FILE,,,"TYPE") ❷            ! determine file type
$   FILE_TYPE = F$EXTRACT(1,F$LENGTH('FILE_TYPE'),FILE_TYPE) ! remove period
$! Remove type from file specification
$   PERIOD_LOC = F$LOCATE(".",FILE)
$   FILE = F$EXTRACT(0,PERIOD_LOC,FILE)
$   ON WARNING THEN GOTO OTHER
$   GOTO 'FILE_TYPE'
$ ELSE                                                            ❸
$   WRITE SYS$OUTPUT FILE, "does not exist"
$ ENDIF                                                           ❹
$!
$ GOTO END
$!
$!
$!
$ FOR:                                                             ❺
$ ON ERROR THEN GOTO PRINT
$ FORTRAN/LIST 'FILE'
$ GOTO LINK
$!
$ PAS:
$   ON ERROR THEN GOTO PRINT
$   PASCAL/LIST 'FILE'
$   GOTO LINK
$!
$ OTHER:
$   WRITE SYS$OUTPUT "Can't handle files of type .''FILE_TYPE'"
$   GOTO END
$!
$ LINK:                                                           ❻
$   ON ERROR THEN GOTO END
$   WRITE SYS$OUTPUT "Successful compilation ...."
$   LINK 'FILE'
$   DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$   RUN 'FILE'
$   GOTO CLEANUP
$!
$ PRINT:                                                          ❼
$   WRITE SYS$OUTPUT "Unsuccessful compilation, printing listing file ...."
$   PRINT 'FILE'
```

```
$!
$ CLEANUP:
$   DELETE 'FILE'.OBJ;
$   DELETE 'FILE'.LIS;
$!
$ END:
$   INQUIRE/NOPUNCTUATION ANS "Process another file (Y or N)?  "
$   IF ANS THEN GOTO TOP
$ EXIT
```

C.13.3 Notes for COMPILE_FILE.COM Command Procedure

❶ The IF command uses the F$SEARCH lexical function to search the directory and determine if the file exists.

❷ The command block following the THEN command:

- Uses the F$LENGTH lexical function to determine the length of the file type
- Determines the file type
- Removes the period from the file type
- Removes the file type from the file specification to determine the file name
- Removes the period from the file name
- Defines an action to perform if an error occurs
- Branches to the label defined by the symbol FILE_TYPE

❸ If the file you entered at the "File to process:" prompt does not exist in the directory, the command following the ELSE command executes.

❹ The ENDIF command ends the IF-THEN-ELSE command language construct.

❺ The procedure compiles the FORTRAN program and branches to the LINK label. If an error occurs during the compilation, the procedure branches to the PRINT label.

❻ The procedure displays that the program compiled correctly, links and runs the program, and branches to the CLEANUP label. The program branches to the END label if an error occurs.

❼ The procedure enters the listing file of the program in the default print queue.

C.13.4 Example: Execution for COMPILE_FILE.COM Command Procedure

```
$ @COMPILE_FILE
File to process: RAND.PAS
Successful compilation
%DELETE-I-FILDEL,WORK:[DESCH]RAND.OBJ;1 deleted (3 blocks)
%DELETE-I-FILDEL,WORK:[DESCH]RAND.LIS;1 deleted (9 blocks)
Process another file (Y or N)? N Return
```

D

Terminal Keys

D.1 LK201 Keyboard

The following table describes how the operating system responds when various keys and control characters are pressed on an LK201 keyboard (VT200 series and later terminals, and workstations). The table assumes that line editing is enabled (the default). Characters not mentioned in the table are treated as null characters.

Character	Hex	System Response
Ctrl/A	01	Switches between overstrike and insert modes
Ctrl/B	02	Recalls previous line
Ctrl/C	03	Interrupts current image (image may define alternate Ctrl/C action)
Ctrl/D	04	Moves cursor left one character
Ctrl/E	05	Moves cursor to end of line
Ctrl/F	06	Moves cursor right one character
Ctrl/H	08	Moves cursor to beginning of line
Ctrl/I	09	Horizontal tab
Ctrl/J	0A	Deletes previous word
Ctrl/M	0D	Line terminator
Ctrl/O	0F	Suspends or resumes echoing of output
Ctrl/Q	11	Resumes output (see Ctrl/S)
Ctrl/R	12	Refreshes current line
Ctrl/S	13	Suspends output (see Ctrl/Q)
Ctrl/T	14	Displays process information (must be enabled with SET CONTROL=T command)
Ctrl/U	15	Deletes characters from cursor to beginning of line

Character	Hex	System Response
Ctrl/V	16	Passes next character or escape sequence to the image without interpreting it as described in this table
Ctrl/X	18	Purges type-ahead buffer; if characters are on the current line, deletes characters from cursor to beginning of line
Ctrl/Y	19	Interrupts current image
Ctrl/Z	1A	Indicates end of file
Data keys	–	Enters appropriate character
<⌫	–	Deletes previous character
Ctrl	–	Modifies another key
Ctrl/[(ESC)	1B	Begins escape sequence
Ctrl/F5	–	Executes answerback message
↓	–	Repeats current line
F1 (No Scroll)	–	Suspends or resumes output
F5 (Break)	–	Shuts down transmission line
F6 (Interrupt)	–	Interrupts current image
F10 (Exit)	–	Terminates current image or command procedure
F12 (Backspace)	08	Moves cursor to beginning of line
F13 (Line Feed)	–	Deletes previous word
F14 (^A)	01	Switches between overstrike and insert modes
←	–	Moves cursor left one character
PFn	–	Can be defined (see Section 3.11)
Return	–	Line terminator
→	–	Moves cursor right one character
Tab	–	Horizontal tab
↑	–	Repeats current line

D.2 VT100 Terminal Series

The following table describes how the operating system responds when various keys and control characters are pressed on VT100 series terminals. The table assumes that line editing is enabled (the default). Characters not mentioned in the table are treated as null characters.

Character	Hex	System Response
Ctrl/A	01	Switches between overstrike and insert modes
Ctrl/B	02	Recalls previous line
Ctrl/C	03	Interrupts current image (image may define alternate Ctrl/C action)
Ctrl/D	04	Moves cursor left one character
Ctrl/E	05	Moves cursor to end of line
Ctrl/F	06	Moves cursor right one character
Ctrl/H	08	Moves cursor to beginning of line
Ctrl/I	09	Horizontal tab
Ctrl/J	0A	Deletes previous word
Ctrl/M	0D	Line terminator
Ctrl/O	0F	Suspends or resumes echoing of output
Ctrl/Q	11	Resumes output (see Ctrl/S)
Ctrl/R	12	Refreshes current line
Ctrl/S	13	Suspends output (see Ctrl/Q)
Ctrl/T	14	Displays process information
Ctrl/U	15	Deletes characters from cursor to beginning of line
Ctrl/V	16	Passes next character or escape sequence to the image without interpreting it as described in this table
Ctrl/X	18	Purges type-ahead buffer; if characters are on the current line, deletes characters from cursor to beginning of line
Ctrl/Y	19	Interrupts current image
Ctrl/Z	1A	Indicates end of file
Data keys	–	Enters appropriate character
Backspace (^H)	08	Moves cursor to beginning of line

Character	Hex	System Response
Break	–	Shuts down transmission line
Ctrl	–	Modifies another key
Ctrl/Break	–	Executes answerback message
Delete	–	Deletes previous character
↓	–	Repeats current line
Esc	1B	Begins escape sequence
←	–	Moves cursor left one character
Line Feed	–	Deletes previous word
No Scroll	–	Suspends or resumes output
PFn	–	Can be defined (see Section 3.11)
Return	–	Line terminator
→	–	Moves cursor right one character
Tab	–	Horizontal tab
↑	–	Repeats current line

Glossary

access control entry (ACE)

An entry in an access control list. Access control entries may specify identifiers and the access rights to be granted or denied to the holders of the identifiers, default protection for directories, or security alarm details.

access control string

A series of 0 to 42 characters that contains login information to be sent to a remote node. On OpenVMS systems, an access control string usually consists of a user name, spaces or tabs, and a password.

account

Every user must have an account to use the system. The account is identified by the user's user name. Different accounts allow different levels of service from the system (for example, the privileges users hold, the times during which they can log in, and so on).

American Standard Code for Information Interchange (ASCII)

A set of 8-bit binary numbers representing the alphabet, punctuation marks, numerals, and other special symbols used in text representation and communications protocol.

assignment statement

In DCL, the association of a symbol name with a character string or numeric value. Symbols can define synonyms for system commands or can be used as variables in command procedures.

batch job

A program that is scheduled and executed under the control of the batch processing subsystem. Control input for a batch job comes from a command procedure stored on disk and output is directed to a disk file.

break-in attempt

An effort made by an unauthorized source to gain access to the system. Because the first system access is achieved through logging in, break-in attempts primarily refer to attempts to log in illegally. These attempts focus on supplying passwords for users

known to have accounts on the system through informed guesses or other trial-and-error methods.

buffer

An internal memory area used for temporary storage of data records during input or output operations.

captive account

A type of OpenVMS account that limits the activities of the user. Typically, the user is restricted to using certain command procedures and commands. For example, the user may not be allowed to use the Ctrl/Y key sequence. This type of account is synonymous with a turnkey or a tied account.

central processing unit (CPU)

The hardware that handles all calculating and routing of input and output as well as executing programs. In short, the CPU is the part of the computer that actually computes.

character string

A contiguous set of bytes. A character string is identified by two attributes: an address and a length. Its address is the address of the byte containing the first character of the string; subsequent characters are stored on bytes on increasing addresses. The length is the number of characters in the string.

close

Terminating all operations on a file.

collating sequence

An order assigned to the characters of a character set (for example, ASCII, Multinational, or EBCDIC) used for sequencing purposes.

command

In DIGITAL Command Language (DCL), an instruction, generally an English word, entered by the user at a terminal or included in a command procedure. A command requests that the software monitoring a terminal or reading a command procedure perform some well-defined activity. For example, entering the COPY command requests that the system copy the contents of one file into another file.

command image

A program associated with and invoked by a DCL command.

command interpreter

A procedure-based system code that executes in supervisor mode in the context of a process to receive, to check the syntax of, and to parse commands entered by the user at a terminal or submitted in a command file.

command level

Input stream for the command interpreter. The initial input stream is always command level 0. An interactive command procedure begins executing at command level 1. A batch job command procedure begins executing at command level 0. You can use the execute procedure (@) command or the CALL command in a command procedure to create up to 32 nested command levels.

command parameter

The positional operand of a command delimited by spaces, such as a file specification, an option, or a constant.

command procedure

A file containing commands and data that the command interpreter can accept. Because command procedures provide a means of automatically passing commands to the operating system, users do not have to manually enter those commands at a terminal. In addition, command procedures permit users to employ such programming techniques as loops, counters, labels, and symbol substitution to set up elaborate command sequences that can be altered through user interaction. Command procedures can also be submitted to the system for processing as batch jobs.

command string

A line (or set of continued lines) containing a command and, optionally, information modifying the command. A command string consists of a command, its qualifiers, its parameters (file specifications, for example), and their qualifiers. A command string is normally terminated by pressing the Return key.

concatenate

The act of linking files together in a series.

CPU

See *central processing unit*.

cursor

An indicator used on a video terminal to point to the screen position where the next character will appear.

data

A general term referring to any representation of facts, concepts, or instructions in a form suitable for communication, interpretation, or processing.

DCL (DIGITAL Command Language)

A command interpreter in an OpenVMS system that provides a means of communication between the user and the operating system.

DECnet/OSI

Family of Digital hardware and software products that implement the Digital Network Architecture (DNA) Phase V, which integrates OSI and DNA protocols. DECnet/OSI is compliant with OSI and compatible with DECnet Phase IV and TCP/IP.

default

A value or operation that is automatically included in a command, unless the user specifies otherwise. In most cases, default settings will be what is ‘‘normal’’ or ‘‘expected.’’

default directory

The directory that the OpenVMS operating system assumes when a directory specification has not been supplied by the user.

default disk

The disk from which the system reads and to which the system writes; by default, all files that you create. The default is used whenever a file specification in a command does not explicitly name a device.

delimiter

A character that separates, terminates, or organizes elements of a character string, statement, or program.

detached process

A process that has no owner. The job controller creates a detached process when a user logs in to the system. It also creates a detached process each time it initiates a batch job or services a request for a logical link connection. Because the job controller does not own the processes it creates, these processes are referred to as detached. The DCL command RUN/UIC and the Create Process system service (specifying a UIC) allow a suitably privileged process to request creation of a detached process.

device

The general name for any peripheral connected to the processor that is capable of receiving, storing, or transmitting data. Card readers, line printers, and terminals are examples of record-oriented devices. Magnetic tape devices and disk devices are examples of mass storage devices. Terminal line interfaces and interprocessor links are examples of communications devices. Devices are not necessarily hardware.

device name

The field in a file specification that identifies the device unit on which a file is stored. Device names also include the mnemonics that identify an I/O peripheral device in a data transfer request. A device name consists of a mnemonic followed by a controller identification letter (if applicable), a unit number (if applicable), and a colon.

DIGITAL Command Language (DCL)

See *DCL (DIGITAL Command Language)*.

directory

A file that briefly catalogs a set of files stored on disk or tape. The directory includes the name, type, and version number of each file in the set, as well as a unique number that identifies the file's actual location and points to a list of its attributes. See also *subdirectory*.

disk

High-speed, random-access devices. There are several kinds of disks. Floppy disks are small, flexible disks. Hard disks are either fixed in place or removable. Removable disk types include a single hard disk enclosed in a protective case and a stacked set of disks enclosed in a protective case.

echo

A terminal-handling characteristic in which the characters typed by the user from the terminal keyboard are displayed on the screen or printer.

editor

A program used to create or modify text in a computer file.

equivalence string

The string associated with a logical name in a logical name table. An equivalence string can be, for example, a device name, another logical name, or a logical name concatenated with a portion of a file specification.

error message

A message sent by the system when some action you have requested fails. Each error message identifies the particular part of the operating system that detected the error. Most error messages result from typing mistakes or mistakes in specifying syntax. Often, you can correct the error by retyping the command correctly.

executable image

An image that can be run in a process. When run, an executable image is read from a file for execution in a process.

expression

Any combination of variables, constants, or both, with operators that the computer can evaluate to produce a result.

field

A set of contiguous bytes in a logical record.

file

A set of data elements arranged in a structure significant to the user. A file is any named and stored program, data, or both, to which the system has access. Access can be of two types: read-only, meaning the file is not to be altered, and read/write, meaning the contents of the file can be altered. See also *volume*.

file name

The field containing a 1- to 39-character name for a file that precedes the file type in a file specification.

file specification

A unique name for a file on mass storage media. It identifies the node, the device, the directory name, the file name, the file type, and the version number under which a file is stored.

file type

The field in a file specification that consists of a period followed by a 0- to 39-character identification. By convention, this field identifies a generic class of files that have the same use or characteristics, such as compiler and assembler listing files, binary object files, and so on.

folder

A subdivision of a file in which you can store mail messages.

foreign command

A symbol that executes an image whose name is not recognized by the command interpreter as a DCL command.

foreign file specification

A file whose specification does not conform to OpenVMS syntax or format.

form feed

The movement of the cursor position to the start of a new page.

full name

Complete specification of a name in the DECdns namespace, including all parent directories in the path from the root directory to the object, directory, or soft link being named; can also include a namespace name, but not necessary when only one namespace exists in a network.

function keys

Keyboard keys that send special signals to the operating system. Function keys are referred to as F*n*, where *n* is the number associated with that key. For example, by pressing F9 in Mail you are telling the system you want to forward a message.

generic device name

A device name that identifies the type of device but not a particular unit; a device name in which the specific controller or unit number is omitted.

global symbol

(1) A symbol defined in a module of a program that is potentially available for reference by another module. The linker resolves (matches references with definitions) global symbols. Contrast with local symbol.

(2) A command language symbol accessible at all command levels.

hardware device

The physical computer equipment, including such mechanical devices as the line printer, the terminals, the mass storage devices, and so forth.

hardcopy terminal

Terminals that print output on paper. See also *terminal*.

help file

A text file in a format suitable for use with the HELP command. Help files can provide up to nine levels of search.

hierarchical directory structure

A structure of directories that has several levels arranged in a tree-like structure, based on a one-to-many relationship.

identifier

An alphanumeric string representing a user or group of users recorded in the rights database and used by the system in checking access requests. There are four types of identifiers: environmental, facility, general, and user identification code (UIC).

image

The procedures and data bound together by the linker to form an executable program. This executable program is executed by the process. There are three types of images: executable, shareable, and system.

indexed sequential file

A record file in which each record has one or more data keys embedded in it. Records in the file are individually accessible by specifying a key associated with the record.

input file

A file containing data to be transferred into the computer.

Often input and output files are confused. DCL usually prompts for these files, but most system utilities require you to identify your input and output files by position in a command line. Be sure of the syntax, or format, for the command you are using.

input stream

The source of commands and date—the user's terminal, the batch stream, or a command procedure.

interactive mode

The mode of communication with the operating system in which you enter a command and the system executes it and responds. One command has to finish executing before you can enter another.

interactive utility

A computer program, invoked with a DCL command, that provides a special environment from which you can perform a specific set of tasks. You work interactively with these utilities by entering subcommands and other information in response to the utility's prompt.

iterative translation

The repetitive translation of a logical name that occurs when a logical name's definition includes another logical name.

job

The accounting unit equivalent to a process and its subprocesses, if any, and all subprocesses that they create. Jobs are classified as batch and interactive. For example, the job controller creates an interactive job to handle a user's requests when the user logs in to the system and it creates a batch job when the symbiont manager passes a command input file to it.

job tree

A hierarchy of all processes and subprocesses, with the main process at the top.

key

(1) In indexed files, a character string, a packed decimal number, a 2- or 4-byte unsigned binary number, or a 2- or 4-byte signed integer within each data record in an indexed file. The user defines the length and location within the records. OpenVMS Record Management Services (RMS) uses the key to build an index.

(2) In relative files, the relative record number of each data record in a data file. OpenVMS Record Management Services (RMS) uses the relative record numbers to identify and access data records in a relative file in random access mode.

(3) In the Sort/Merge utility, the data field in a record that contains the information by which the user wants to sort the records.

keyboard

An input device that can be operated similarly to a typewriter.

keypad

The small set of keys next to the main keyboard on a terminal.

keyword

A word reserved for use in certain specified syntax formats, usually in a command string or a statement.

lexical function

A command language construct that the DIGITAL Command Language (DCL) command interpreter evaluates and substitutes before it parses a command string. Lexical functions return information about the current process (the user identification code (UIC) or default directory, for example) and about character strings (their length or the location of substrings, for example).

line editor

A program that allows you to make additions and deletions to a file line by line.

line printer

An output device that prints files one line at a time. It is used for printing large amounts of output that would otherwise tie up a slower device. Almost every system has a device designated as the line printer. In some cases, the ‘‘line printer’’ is actually a high-speed terminal.

local node

The network node at which the user is physically located.

local symbol

(1) A symbol meaningful only to the module that defines it. Symbols not identified to a language processor as global symbols are considered to be local symbols. A language processor resolves (matches references with definitions) local symbols. They are not known to the linker and cannot be made available to another object module. They can, however, be passed through the linker to the debugger. Contrast with global symbol.

(2) A command language symbol name that is accessible only at the current command level and subsequently invoked levels. It is deleted when the command level at which it is defined exits.

logging in

The identification of a user to the system. When users log in, they type a user name and password in response to prompts from the system. If the user name and the password match an account on the system, the user is allowed access to the system.

logging out

The process of entering the DIGITAL Command Language (DCL) command LOGOUT, which informs the operating system that the user has finished using a particular terminal.

logical device name

A character string that equates a somewhat cryptic device name to a short, meaningful name.

logical expression

An expression that has a true or false value.

logical name

A user-specified name for any portion or all of a file specification. For example, the logical name INPUT can be assigned to a terminal device from which a program reads data entered by a user. Logical name assignments are maintained in logical name tables for each process, each group, and the system. Logical names can be assigned translation attributes, such as terminal and concealed.

logical name table

A table that contains a set of logical names and their equivalence strings for a particular process, a particular group, or the system.

login class

User's method of logging in to the system. System managers can control system access based on the login class: local, dialup, remote, batch, or network.

login command procedure

A command procedure that is automatically executed at login and at the beginning of a batch job.

login directory

The default directory established by LOGINOUT when a user logs in.

magnetic tape

A medium on which data can be stored and accessed.

mass storage device

An input/output device on which data and other types of files are stored while they are not being used. Typical mass storage devices include disks, magnetic tapes, and floppy disks.

memory

A series of physical locations into which data or instructions can be placed in the form of binary words. Each location in memory can be addressed and its contents can be altered. Memory should not be confused with mass storage devices.

network

A collection of interconnected, individual computer systems.

node

(1) An individual computer system in a network that can communicate with other computer systems in the network.

(2) On OpenVMS VAX systems, a VAXBI interface—such as a central processing unit, controller, or memory subsystem—that occupies one of 16 logical locations on a VAXBI bus.

(3) On OpenVMS VAX systems, a VAX processor or HSC that is recognized by system communications services (SCS) software.

node specification

The first field in a file specification. This field identifies the location of a computer system in a network.

null value

A string with no characters that is represented in a command procedure by two quotation marks (" ").

numeric expression

A mathematical statement consisting of a collection of operands connected by arithmetic operators.

object

A passive repository of information to which the system controls access. Access to an object implies access to the information it contains.

open

The act of preparing a data set or file for processing.

open account

An account that does not require a password.

operand

The part of an expression that contains a value. Operands are acted on by operators during expression evaluation to produce a result.

operating system

An integrated collection of programs that controls the execution of computer programs and performs system functions.

operator

The part of an expression that tells the computer how to manipulate the operands. For example, the plus sign (+) is an operator that tells the computer to perform addition.

output file

A file that contains the results of a processing operation; for example, a file that has been sorted or edited.

parameter

(1) A value passed to a command procedure equated to a symbol ranging from P1 through P8. See also *command parameter*.

(2) An entry in the volatile or permanent database for a network management component.

parsing

(1) Breaking a command string into its elements to interpret it.

(2) Interpreting a file specification, as is done by OpenVMS Record Management Services (RMS).

password

A character string that users provide at login time to validate their identities and as a form of proof of their authorization to access their accounts. There are two kinds of passwords—system passwords and user passwords. User passwords include both primary and secondary passwords.

physical device name

A character string that uniquely identifies a physical device (such as a storage disk or a terminal) to the system.

primary password

A type of user password that is the first user password the system requests from the user. Systems may optionally require a secondary password as well. The primary password must be the password that is associated with the user name.

print form

A set of attributes that defines page set up and stock for printing.

print queue

A list of files waiting to be printed.

priority

A rank assigned to a process to determine its precedence in obtaining system resources when the process is running.

private volume

A volume that has been allocated by a process for its own exclusive use.

process

The basic entity scheduled by the system software. A process provides the context in which an image executes. A process consists of an address space and both hardware and software context.

program

A series of instructions aimed at a particular result. Programming languages are a means of describing procedures so that they can be performed by a computer. See also *image*.

program stub

A temporary section of code that is used during the testing phase of writing command procedures. A program stub usually outputs a message stating the procedure it is replacing.

prompt

A character string appearing on a terminal screen indicating that the user must provide input.

protected object

An object containing shareable information to which the system controls access. See also *object*.

protection code

A series of letters that specify what access different categories of system users can have to a file or to another protected object and what they can do to it when they access it.

proxy login

A type of login that permits a user from a remote node to effectively log in to a local node as if the user owned an account on the local node. However, the user does not specify a password in the access control string. The remote user may own the account or share the account with other users.

qualifier

A portion of a command string that modifies a command verb or command parameter by selecting one of several options. A qualifier, if present, follows the command verb or parameter to which it applies and is in the format /qualifier[=option]. For example, in the command string ''PRINT filename /COPIES=3,'' the COPIES qualifier indicates that the user wants three copies of a given file printed.

queue

(1) A line of jobs to be processed; for example, a batch job queue or a printer job queue. Processing occurs primarily in first-in/first-out (FIFO) order, but does reflect the priority of the process that submitted the job. See also *print queue*.

(2) To add an entry in a list or table, often by using the INSQUE instruction.

random access

A method for retrieving or writing data in which the location of the data to be retrieved or written is not dependent on the location of previously retrieved or written data. Random access refers to memory or mass storage devices on which all information is equally accessible.

read

The act or capability of an image to accept data. For example, when a TYPE command is issued, the system reads the designated file from disk and writes it to the terminal. See also *write*.

record file address (RFA)

The unique address of a record in a file. The RFA allows previously accessed records to be accessed randomly at a subsequent time. This access occurs regardless of the file organization.

Record Management Services (RMS)

See *RMS*.

record-oriented device

A device such as a terminal, line printer, or card reader. A record-oriented device's physical record is the largest unit of data that a program can access in one I/O operation.

record sorting

A sorting process in which records are kept intact and an output file consisting of complete records is produced.

relative file organization

The arrangement of records in a file in which each record occupies a cell of equal length within a bucket. Each cell is assigned a successive number, which represents its position relative to the beginning of the file.

remote node

Any node in a network, other than the node that you are currently logged in to.

restricted account

A type of OpenVMS account with a secure login procedure. The user is not allowed to use the Ctrl/Y key sequence during the system or process login command procedure. Control may be turned over to the user following execution of the login command procedures.

reverse video

A feature of a video terminal that reverses the default video contrast. If the default display is black figures on a white background, reverse video displays white figures on a black background.

RMS

A set of operating system procedures that are called by programs to process files and records within files. RMS allows programs to issue GET and PUT requests at the record level (record I/O) as well as read and write blocks (block I/O). RMS is an integral part of the system software; its procedures run in executive mode.

scrolling

A feature of a video terminal that allows the display of more than one screen of text by vertical movement. For example, when the TYPE command is entered, new output appears at the bottom of the screen as the oldest output disappears off the top.

secondary password

A user password that may be required at login time immediately after the primary password has been submitted correctly. Primary and secondary passwords can be known by separate users to ensure that more than one user is present at the login. A less common use is to require a secondary password as a means of increasing the password length so that the total number of combinations of characters makes password guessing more time-consuming and difficult.

secure terminal server

OpenVMS software designed to ensure that users can log in only to terminals that are already logged out. When the user presses the Break key on a terminal, the secure terminal server (if enabled) responds by first disconnecting any logged-in process and then initiating a login. If no process is logged in at the terminal, the login can proceed immediately.

sequential access mode

The retrieval or storage of records in which a program reads or writes records one after the other in the order in which they appear, starting and ending at any arbitrary point in the file.

sequential file organization

A file organization in which records appear in the order in which they were originally written. The records can be fixed or variable length. Records can be accessed sequentially or randomly by record address. Fixed length records can also be accessed randomly by relative record number.

software

The collection of images, procedures, rules, and documentation associated with the operation of a particular computer system. For example, the operating system is software.

specification file

A command file used in the Sort/Merge utility to specify the commands and qualifiers needed to complete a sort operation.

string

A connected sequence of characters. When a text editor searches for a word or phrase in a text file, it is looking for a string. The character sequence that forms a command is often called a command string.

subdirectory

A directory file, cataloged in a higher level directory, that lists additional files belonging to the owner of the directory.

subprocess

A subsidiary process created by another process. The process that creates a subprocess is its owner. A process and its subprocesses share a pool of quotas and limits. When an owner process is removed from the system, all its subprocesses (and their subprocesses) are also removed.

subroutine

A subsidiary routine that executes when called by another program. A subroutine is often called repeatedly until a certain condition is met.

symbol

An entity that, when defined, represents a particular function or entity (for example, a command string, directory name, or file name) in a particular context.

symbol scope

The set of command procedure levels from within which the symbol can be accessed.

syntax

The particular form of a command, including the spelling and the order of qualifiers and parameters. Misspelled words are the most common syntax errors.

system manager

The person who makes resources available to users and sets up restrictions governing the use of such resources.

system password

A password required by a terminal before login can be initiated.

terminal

The general name for a peripheral device that has a keyboard and a video screen or printer. Under program control, a terminal enables users to type commands and data from the keyboard and receive messages on the video screen or printer. Examples of terminals are the LA36 DECwriter hardcopy terminal, the VT100 video display terminal, and the VT240 series video terminal.

timeout

The expiration of the time limit during which a device is to complete an I/O transfer.

time-stamp

A text string that fully specifies a data and time. For example, 11-DEC-1995 17:13:21.

UAF (user authorization file)

The file that holds details of each account on the system. The UAF contains the user name, password, user identification code (UIC), quotas, limits, and privileges assigned to each account.

UFD (user file directory)

A file that briefly catalogs a set of files stored on disk or tape. The UFD includes the name, type, and version number of each file in the set. It also contains a unique number that identifies that file's actual location and points to a list of its file attributes. See also *directory*.

UIC (user identification code)

The pair of numbers assigned to users, files, global sections, command event flag clusters, and mailboxes. The UIC specifies the type of access (read, write, or read/write, and in the case of files, execute, delete, or both) available to the owner, group, world, and system.

user authorization file (UAF)

See *UAF (user authorization file)*.

user password

A password that is associated with a user. This password must be correctly supplied when the user attempts to log in so that the user is approved for access to the system. The two types of user passwords are primary and secondary; the terms also represent the sequence in which they are entered.

utility

A program that provides a set of related general-purpose functions, such as a program development utility (an editor, a linker), a file management utility (file copy or file format translation program), or an operations management utility (disk quotas, diagnostic program).

version number

The numeric component of a file specification. When a file is edited, its version number is increased by one.

video terminal

A terminal with a video screen for accepting output. See also *terminal*.

volume

A mass storage media such as a disk pack or reel of magnetic tape. The volume is the largest logical unit of the file structure.

volume set

The file-structured collection of data residing on one or more mass storage media.

wildcard character

A nonalphanumeric character such as an asterisk (*) or percent sign (%) that is used within, or in place of, a file name, a file type, a directory name, or a version number in a file specification to indicate ‘‘all’’ for the given field.

write

The act or capability of an image to send data. For example, when a PRINT command is issued, the specified file is read from wherever it is stored and is written to the printer. See also *read*.

Index

A

Absolute time
  combined with delta time, 3–13
  default values, 3–11
  rules for entering, 3–11
  specifying keywords, 3–12
  syntax, 3–11
Access control
  object security profiles, 19–3
Access control strings, 4–6, 19–7
  copying files between nodes with, 4–13
  format, 19–7
  format in node names, 4–8
  overriding, 13–5
  protecting information in, 19–7
  rules for entering, 4–8
  within command procedures, 19–7
Access list, 19–5
Access modes, 13–9, 13–10
  displaying, 13–21
  executive, 13–10
  user, 13–10
Access types
  and security audit, 19–11
Accounting information
  obtaining during logout, 2–24
Accounts
  auditing access, 19–10
  expiration of, 2–16
  first login, 2–4
  initial password, 2–4
  open, 2–7
  renewing expired, 2–17
  restricted, 2–7
  secondary password, 2–5
  types of, 2–6
ACEs (access control entries), 19–12
ACLs (access control lists)
  default protection, 19–6
  protecting files, 4–16
Alarms
  enabling for security, 19–10
  events triggering, 19–10
ALLOCATE command, 12–10
  format, 12–3
  /GENERIC qualifier, 12–4
Allocating
  a device by logical name, 12–4
  a specific device type, 12–4
  class fields, 12–10
  disk drives, 12–3
  first available device, 12–3
  generic devices, 12–4
  physical device, 12–3
  tape drives, 12–3
Ampersand (&)
  in symbol definitions, 14–32 to 14–33
Apostrophe (')
  in symbol definitions, 14–31, 14–32
Arrow keys
  moving cursor within DCL command line, 3–19
ASCII (American Standard Code for Information Interchange)
  ''a'' character set, 4–11, B–1
  character set, B–1
  full character set, B–2
ASSIGN command, 13–3
  access modes, 13–9
Assignment statement
  creating symbols, 14–3
Asterisk (*)
  as EDT prompt, 9–3
  as wildcard character
    in directory specifications, 4–9
    in UIC format directory specifications, 5–10
    rules for using, 4–9
  in symbol definitions, 14–4
At sign (@)
  as execute procedure command, 15–21
  using with a distribution list in Mail, 6–7, 6–8

B

Backspace key, 3–17, 3–19
Batch jobs
  affected by shift restrictions, 2–12
  authorization, 2–10
  changing characteristics, 18–16
  changing entries, 18–16
  definition, 11–13

Batch jobs (cont'd)
deleting, 18–19
displaying status, 18–18
job entries, 18–10
log files, 18–14
output, 18–14
providing input to, 18–13
restarting, 18–19, 18–20
specifying multiple command procedures, 18–12
specifying start time, 18–11
stopping, 18–19
SUBMIT command qualifiers, 18–17
submitting, 18–10
command procedures, 15–24, 18–10
synchronizing multiple procedures, 18–20, 18–21
uses of, 18–10
BATCH$RESTART symbol
using in command procedures, 15–25
BATCH.COM command procedure, C–27
sample execution, C–31
Bell character, 2–5
Break-in attempts, 2–12
auditing, 19–10
evasion, 2–12
Buffers
recall DCL command, 3–14

C

CALC.COM command procedure, C–25
sample execution, C–27
Case sensitivity
in DCL command lines, 3–2
using SET FIND CASE EXACT command, 8–32
with REPLACE command in EVE, 8–32
Character sets
ASCII, B–1
DEC Multinational, B–3
Character strings
and symbols, 14–8
as literal text, 16–9
comparing, 14–11, 14–12
converting to integer values, 14–25, 14–26
converting using lexical functions, 17–17
defining, 14–9
evaluation of, 14–24
examining with lexical functions, 17–13
expressions
in symbols, 14–9
operands, 14–10
extracting with lexical functions, 17–13, 17–14
forcing symbol substitutions in, 16–10
operations, 14–10
Character strings (cont'd)
quotation marks (" ") in, 16–10
replacing, 14–13
with lexical functions, 17–12
Characters
alphanumeric, 14–8
nonprintable, 14–8
special, 14–8
Clean-up operations, 15–18
Clean-up tasks
in command procedures, 15–16
CLEANUP.COM command procedure, 15–19
Combination time
rules for entering, 3–13
syntax, 3–13
Command interpreter
commands performed in, 16–11
Command levels
definition, 15–29
with SET [NO]ON command, 15–34
Command lines
deleting characters, 3–17
editing, 3–16
in Mail, 6–3
in Phone, 7–3
including in command procedures, 15–3
recalling, 3–14
wrapping, 3–17
Command procedures
access control strings, 19–7
assigning variables, 15–9
BATCH.COM, C–27
CALC.COM, C–25
canceling with Ctrl/Y, 15–35
changing verification settings, 17–4
clean-up tasks, 15–16, 15–18
CLEANUP.COM, 15–19
closing files, 15–17
comments, 15–5
COMPILE_FILE.COM, C–31
completing, 15–19
CONVERT.COM, C–1
creating, 15–3
debugging, 15–14
SET PREFIX command, 15–15
SET VERIFY command, 15–15
SHOW SYMBOL command, 15–15
default file type of, 15–3
definition, 15–1
deleting files, 15–17
designing, 15–8
detecting errors, 15–41
determining conditionals, 15–8
determining variables, 15–8
DIR.COM, C–9
disabling verification settings, 17–4
dollar signs ($), 15–4
duplicate labels, 15–5

Command procedures (cont'd)
EDITALL.COM, C–15
enabling verification during execution, 15–16
ending, 15–13
error handling, 15–30
SET NOON command, 15–34
executing, 15–21
as batch jobs, 15–24
as remote batch jobs, 15–25
from within other command procedures, 15–21
interactively, 15–23
on private disks, 15–27
on remote nodes, 15–21
on tape volumes, 15–27
redirecting interactive output, 15–24
restarting batch jobs, 15–25
with automatic foreign commands, 14–38
without using symbols, 14–38
EXIT command, 15–13, 15–29
exiting, 15–29
FORTUSER.COM, C–19
GETPARMS.COM, C–13
handling label errors, 15–30
IF command, 15–10
including command lines, 15–3
including DCL commands, 15–3
including literal characters, 15–10
input, 16–2
interrupting with Ctrl/Y, 15–35
labels, 15–4
LISTER.COM, C–23
login, 15–45
LOGIN.COM, 18–11, 18–12, 18–14
MAILEDIT.COM, C–17
nested
passing data to with parameters, 16–5
order of DCL commands, 15–9
output, 16–9
passing data to with parameters, 16–3
personal login, 15–45
in captive accounts, 15–46
program stubs, 15–10
prompting for user input in, 16–6
redefining SYS$INPUT for, 16–5
REMINDER.COM, C–5
replacing program stubs, 15–19
restoring defaults, 15–18
saving defaults, 15–18
specifying in batch job, 18–12
steps for writing, 15–7
STOP command, 15–14
SYLOGIN.COM, 18–10
SYS.COM, C–11
testing, 15–14
testing conditionals, 15–9
Command procedures (cont'd)
THEN command, 15–10
types, 15–1
using variables, 17–12
writing, 15–7
Command qualifiers, 3–9
Commands
abbreviating, 3–7
abbreviating in command procedures, 3–7
canceling, 3–5
date formats, 3–11
entering in Mail, 6–2
time formats, 3–11
Comments
in command procedures, 15–5
in distribution lists, 6–7
use of exclamation point (!), 6–7
COMPILE_FILE.COM command procedure, C–31
sample execution, C–34
Condition codes
definition, 15–41
displaying, 15–41
severity levels of, 15–43
with EXIT command, 15–42
Conditionals
definition, 15–7
determining in command procedures, 15–8
testing in command procedures, 15–9
CONNECT command
/CONTINUE qualifier, 18–8
CONTINUE command, 18–4
Control characters
ASCII values, D–3
Controller designation field
default value, 12–10
definition, 12–10
Conversation text
in Phone, 7–3
CONVERT.COM command procedure, C–1
sample execution, C–5
COPY command, 4–12
in Mail, 6–13
printing from a private volume, 12–2
/SINCE qualifier, 4–12
Copying files, 4–12
between nodes, 4–12
restrictions when using Mail, 4–13
using Mail, 4–13
CREATE command, 4–11
/DIRECTORY qualifier, 5–3, 5–7
Ctrl keys, 3–18
common, 3–2
Ctrl/A key sequence, 3–19
Ctrl/B key sequence, 3–14, 19–7
recalling commands with, 3–14, 3–19

Ctrl/C key sequence, 3–18
canceling Mail messages, 6–5, 6–10
Ctrl/E key sequence, 3–19
Ctrl/T key sequence
checking the status of processes, 2–21
interrupting DCL commands with, 3–18
Ctrl/U key sequence, 3–17
Ctrl/W key sequence
refreshing screen displays with, 18–4
Ctrl/Y key sequence
aborting remote sessions with, 2–24
canceling command procedures, 15–35
canceling DCL commands, 3–18
disabling, 15–40
effects of entering, 15–37
enabling, 15–40
exiting, 15–36
exiting from loops when disabled, 15–40
interrupting command procedures, 15–35
interrupting DCL commands, 3–18
interrupting images with, 18–4
resuming an EDT editing session, 9–9
setting action routines, 15–36
Ctrl/Z key sequence
as end-of-file terminator, 3–18, 4–11
sending files in Mail, 6–10
sending Mail messages, 6–5

D

Data
including in command procedures, 16–2
logical, 14–18, 14–19, 14–20
writing to terminals, 16–9
writing using the WRITE command, 16–9
Data files
including programs in, 16–8
Data types
using lexical functions with, 17–17
Dates
specifying absolute and delta date and time combinations, 3–13
specifying absolute date and time, 3–11
specifying delta date and time, 3–12
DCL (DIGITAL Command Language)
definition, 3–1
DCL commands
abbreviating, 3–7
in command procedures, 3–7
abbreviating qualifiers, 3–9
automatic foreign commands, 14–39
canceling, 3–18
case sensitivity, 3–2
command qualifiers, 3–9
conflicting qualifiers, 3–10
constructing, 3–3
default qualifiers, 3–9
defaults, 3–5
DCL commands (cont'd)
editing command line, 3–16
entering, 3–2
including in command procedures, 15–3
interrupting, 3–18
lowercase letters, 3–6
maximum number of elements, 3–7
multiple line command, 3–6
order of in command procedures, 15–9
parameter qualifiers, 3–9
parameters, 3–5
performed in the command interpreter, 16–11
positional qualifiers, 3–9
prompt, 3–3
recalling, 3–19
with Down arrow key, 3–19
redirecting output from, 16–10
required punctuation, 3–7
required spaces, 3–6
rules for entering, 3–6
specifying parameters, 3–8
syntax, 3–3, 3–4
uppercase letters, 3–6
use of null characters, 3–7
verbs, 3–3
.DDIF files
in Mail, 6–9
DEALLOCATE command, 12–3
DEASSIGN command, 13–22, 13–23
Debugging
command procedures, 15–14
SET PREFIX command, 15–15
SET VERIFY command, 15–15
SHOW SYMBOL command, 15–15
DEC Multinational character set, B–1, B–3
DEC Text Processing Utility (DECTPU), 8–1
DECnet
losing connection to remote system, 2–25
manipulating files with, 4–12
specifying full node names, 4–6
Defaults
definition, 2–19
file protection, 19–5
for DCL commands, 3–5
restoring from within command procedures, 15–18
saving from within command procedures, 15–18
values provided by system, 3–5
DEFINE command, 13–3, 13–4
access modes, 13–9, 13–10
/KEY qualifier in Mail, 6–21
in initialization file, 6–22
DELETE command, 4–15
/ENTRY qualifier, 18–19
in Mail, 6–16

DELETE command (cont'd)
using with F$SEARCH lexical function, 17–11
Delete key, 3–17, 3–19
Deleting
characters in command line, 3–17
Delta time
combined with absolute time, 3–13
default values, 3–12
rules for entering, 3–12
syntax, 3–12
Device field, 4–2
Device names
generic, 12–10
logical, 12–9
specifying in a command, 12–9
VMScluster, 12–10
Devices, 12–1
accessing, 12–1
accessing files, 12–8
accessing volume sets, 12–8
allocating, 12–3
by device type, 12–4
by logical name, 12–4
command format, 12–3
first available, 12–3
generic devices, 12–4
physical device, 12–3
dismounting allocated, 12–11
displaying information, 12–2
identifying, 12–8
obtaining information about using F$DEVI lexical function, 17–10
private, 12–8
security, 12–2
specifying physical device name, 12–8
volumes, 12–2
Volumes
private, 12–2
Dialup lines
controlling access to, 2–5
Dialups
breaking connections, 2–26
login failures, 2–12
DIR.COM command procedure, C–9
sample execution, C–11
Directories
access, 5–8
definition, 5–1
deleting, 5–4
DIRECTORY command, 5–3
in Mail, 6–4, 6–14
Directory field
definition, 4–2
using a percent sign (%) wildcard character in, 4–10
using an asterisk (*) wildcard character in, 4–9
Directory files
creating, 5–3
default, 5–5
format, 5–3
protection, 5–7
setting default to another, 5–5
top-level, 5–1
Directory names
format in file specifications, 5–3
replacing
with ellipsis (...) wildcard character, 5–8
with hyphen (-) wildcard character, 5–9
translating UIC format to named format, 5–10
Directory specifications
definition, 5–3
format, 5–3
rules for entering, 5–3
Directory structures
sample, 5–1
.DIS file type
with distribution lists, 6–8
Disabling error checking, 15–34
Disks
deallocating drives, 12–3
mounting, 12–7
DISMOUNT command, 12–11
/NOUNLOAD qualifier, 12–11
Dismounting
volumes, 12–11
allocated devices, 12–11
Displaying files in directories, 5–3
Distribution lists
creating with an editor, 6–7
sending mail to from DCL level, 6–8
using in Mail, 6–8
Dollar sign ($)
as DCL prompt, 2–2, 3–2
in command procedures, 15–4
in file names, 4–2
in VMScluster device specifications, 12–10
Down arrow key
recalling commands with, 3–14, 3–19
DSR (DIGITAL Standard Runoff)
commands
for flag recognition, 10–20
summary, 10–11 to 10–22
creating a file, 10–2
creating a table of contents, 10–5
defaults, 10–5
qualifiers, 10–6
tailoring, 10–6, 10–7
creating an index, 10–8
defaults, 10–8
qualifiers, 10–9

DSR (DIGITAL Standard Runoff)
creating an index (cont'd)
tailoring, 10–9
description, 10–1
disabling defaults, 10–5
formatting
appendix, 10–17
chapter, 10–17
emphasizing text, 10–15
figures, 10–16
filling text, 10–13
footnotes, 10–17
horizontal spacing, 10–14
index, 10–19
justifying text, 10–13
lists, 10–16
margins, 10–13
notes, 10–17
page size, 10–11
paging, 10–12 to 10–13
paragraphs, 10–15
running heads, 10–11
section, 10–18
table of contents, 10–19
vertical spacing, 10–14
formatting a file, 10–2
processing a file, 10–2
RUNOFF command, 10–2
overriding, 10–3
qualifiers, 10–3
using defaults, 10–4

E

EDIT command, 9–2
/EDT qualifier
/READ_ONLY qualifier, 4–14
/TPU qualifier
/READ_ONLY qualifier, 4–14
EDITALL.COM command procedure, C–15
sample execution, C–17
Editing command lines, 3–16
deleting characters, 3–17
enabling line editing, 3–16
insert mode, 3–16
overstrike mode, 3–16
Editing files
using EDT, 9–1
EDIT/TPU command
to invoke EVE, 8–4
EDT editor
buffer definition, 9–2
canceling commands, 9–5
changing case of text
summary of commands and keys, 9–16
changing modes, 9–2, 9–7, 9–8
summary of commands and keys, 9–9
commands
summary of processing commands and keys, 9–19
creating files, 4–11, 9–2
defining keys
summary of commands and keys, 9–17
definition, 9–1
deleting text
summary of commands and keys, 9–12
displaying files with, 9–3
editing existing files, 9–2
help
keypad editing, 9–6
line mode, 9–6
nokeypad mode, 9–7
indenting text
summary of commands and keys, 9–15
inserting text
summary of commands and keys, 9–11
invoking, 9–2
journal files, 9–9
keypad commands, 9–5
keypad editing, 9–1, 9–5
help, 9–6
keypad mode, 9–7, 9–8
entering line-editing commands, 9–8
line editing, 9–1
line mode, 9–7, 9–8
help, 9–6
line-editing commands, 9–3
locating text
summary of commands and keys, 9–13
modifying files, 4–11
moving text
summary of commands and keys, 9–15
moving the cursor
summary of commands and keys, 9–10
multiple buffers
summary of commands and keys, 9–17
nokeypad editing, 9–2
nokeypad mode
help, 9–7
recovering from Ctrl/Y, 9–9
restoring text
summary of commands and keys, 9–12
restoring the display, 9–9
saving edits, 9–7
screen settings
summary of commands and keys, 9–18
specifying a range, 9–3
substituting text
summary of commands and keys, 9–14
summaries of commands and keys, 9–9 to 9–21
terminal settings
summary of commands and keys, 9–18
terminating a session, 9–7

EDT editor (cont'd)
using line numbers, 9–3
Ellipsis (...)
wildcard character in directory names, 5–8, 5–9
Environmental identifiers
example, 19–3
Equivalence strings, 13–1
defining multiple logical names for, 13–7
Error handling
SET NOON command, 15–34
Errors
detecting in command procedures, 15–41
during login, 2–3
handling in command procedures, 15–30
label, 15–30
EVE
as default editor, 8–2
Mail, 6–19
attributes
disabling prompting, A–17
saving, A–10 to A–19, A–20, A–34, A–36, A–39
a section file, A–38 to A–39
in a command file, A–19, A–20, A–40
setting, A–10, A–13
box editing, 8–23
commands, 8–24, 8–25
cutting text, 8–24
effects of pending delete, 8–26
insert mode, 8–26
overstrike mode, 8–26
pasting text, 8–24
restoring text, 8–26
selecting text, 8–23
tutorial, 8–25
buffer-change journaling, 8–37
commands, 8–38
definition, 8–37
disabling, 8–37, 8–40
enabling, 8–41
file names, 8–38, 8–39
files, 8–38
RECOVER BUFFER command, 8–39
/RECOVER qualifier, 8–39
recovering edits, 8–39, 8–40
buffers, 8–44
BUFFER command, 8–48
changing status, 8–47
creating, 8–44
definition, 8–44
deleting, 8–44, 8–46 to 8–47
displaying information, 8–46
displaying messages, 8–48
editing two buffers, 8–51
GET FILE command, 8–48
manipulating, 8–44
EVE
buffers (cont'd)
multiple, 8–48
OPEN SELECTED command, 8–48
SET BUFFER command, 8–47
splitting windows, 8–51
viewing two sections, 8–51
changing modes, 8–14
character-cell terminals, 8–2
command files, A–26, A–34
adding functions to EVE, A–27
converting to section files, A–27
rules for writing, A–28, A–29, A–30
setting default, A–40
setting editing defaults, A–27
startup, A–41
commands
DEFINE KEY, A–3
EXIT, 8–8
FIND, 8–28
formatting, 8–41
HELP, 8–3
LEARN, A–9
QUIT, 8–8
REPLACE, 8–31, 8–32
RESTORE BOX SELECTION, 8–26
RESTORE SELECTION, 8–26
SET FIND CASE EXACT, 8–29, 8–32
SET PENDING DELETE, 8–26
SPAWN, 8–52, 8–53
text formatting, 8–42
WRITE FILE, 8–8
copying text, 8–22
tutorial, 8–23
creating a subprocess, 8–52
creating files, 4–11
default settings, A–10
buffer-specific, A–12
direction, A–13
defining keys, A–2, A–3
abbreviations, A–4
for EVE commands, A–3
definition, 8–1
editing keys, 8–41
EDIT/TPU command line qualifiers, 8–33
for modifying buffers, 8–35
for overriding /READ_ONLY or /NOWRITE, 8–35
for start position, 8–33 to 8–34
for work files, 8–34 to 8–35
emulating EDT, A–41
defining keys, A–41, A–42
replacing EDT macros with DECTPU procedures, A–43
replacing EDT nokeypad statements with DECTPU procedures, A–44
setting case sensitivity, A–43
setting cursor, A–42

EVE
emulating EDT (cont'd)
setting right margin, A–42
setting scroll margins, A–42
using SET KEYPAD EDT command, A–41
using the WPS keypad ruler key, A–44, A–45
entering commands, 8–5
with defined keys, 8–6
entering text, 8–13
commands, 8–14
editing keys, 8–13
including files, 8–13
insert mode, 8–14
overstrike mode, 8–14
special characters, 8–13
tutorial, 8–15
erasing text, 8–16
commands, 8–17
editing keys, 8–16
tutorial, 8–18
with pending delete, 8–26
exiting, 8–7, 8–8
file backups, 8–37
help, 8–3
accessing with keypad, 8–3
HELP command, 8–3
initialization files, A–14, A–32, A–34
assigning EVE commands, A–3, A–4
commands that define the environment, A–33
creating, A–32, A–33
default, A–33
defining the GOLD key, A–9
specifying, A–33
insert mode, 8–14
box editing, 8–26
invoking, 8–4
from search lists, 8–35
with multiple input files, 8–35, 8–37
with wildcard directory names, 8–35, 8–36
with wildcards, 8–35, 8–36
key name conventions, 8–1
keys
defined
VT100 series terminals, 8–7
VT200 series terminals, 8–6
VT300 series terminals, 8–6
VT400 series terminals, 8–6
defining
control keys, A–5
GOLD key, A–8, A–9
differences between EVE and DECTPU, A–5
GOLD key, A–7
EVE
keys (cont'd)
names and labels, A–5
removing GOLD key definitions, A–9
undefinable, A–6
learn sequences, A–2, A–9
defining, A–9
tutorial, A–10
locating text, 8–27
commands, 8–27
exact case, 8–29
marking locations, 8–31
search direction, 8–28
using wildcards, 8–30
with FIND command, 8–28
modifying files, 4–11
moving text, 8–19
commands, 8–21
editing keys, 8–20
tutorial, 8–22
moving the cursor, 8–8
commands, 8–10
keys, 8–9
tutorial, 8–12
overstrike mode, 8–14
box editing, 8–26
quitting a session, 8–8
reading batch job log files with, 18–14
reading files, 8–49
replacing text, 8–31
restoring text, 8–16
after a pending delete operation, 8–26
box editing, 8–26
commands, 8–17
editing keys, 8–16
tutorial, 8–18
saving edits, 8–7
with EXIT command, 8–8
with WRITE FILE command, 8–8
section file, A–10, A–13, A–17, A–18, A–25, A–26, A–34
default, A–26
modifying, A–25
specifying, A–18
startup, A–25
spawning a subprocess, 8–52, 8–53
startup files, A–1, A–34
summary of features, 8–2
switching between EVE and DCL, 8–52
tailoring the standard editor, A–2
using DECTPU, A–20
compiling procedures, A–24
creating command files, A–21
debugging package, A–21
entering commands, A–22
entering statements, A–22
specifying debug file, A–21

EVE
using DECTPU (cont'd)
writing command procedures, A–22, A–23
using in Mail, 6–18
windows, 8–49
commands, 8–50
keys, 8–50
workstations, 8–2
writing files, 8–49
Exclamation point (!)
in distribution lists, 6–7
Execute procedure (@) command, 15–21
Executing command procedures, 15–21
as batch jobs, 15–24
as remote batch jobs, 15–25
from within other command procedures, 15–21
interactively, 15–23
on private disks, 15–27
on remote nodes, 15–21
on tape volumes, 15–27
redirecting interactive output, 15–24
restarting batch jobs, 15–25
with automatic foreign commands, 14–38
without using symbols, 14–38
EXIT command
in command procedures, 15–13
in Mail, 6–3
using with command procedures, 15–29
when to use, 15–14
with condition codes, 15–42
Exiting
from command procedures, 15–29
Expressions
and symbols, 14–8
converting value data types in, 14–25
DCL evaluations of, 14–24
logical, 14–19, 14–20
precedence of operators, 14–23
EXTRACT command
in Mail, 6–10

F

F$CONTEXT lexical function, 17–8, 17–9
F$CVTIME lexical function, 17–13, 17–14
F$DIRECTORY lexical function, 14–23
F$ELEMENT lexical function, 17–13, 17–14
F$ENVIRONMENT lexical function
changing default file protections with, 17–5
F$EXTRACT lexical function, 17–13, 17–14
F$FAO lexical function, 17–15
F$GETQUI lexical function
obtaining queue information, 17–7
F$GETSYI lexical function
obtaining information
system, 17–6, 17–7
VMScluster system, 17–6, 17–7
F$INTEGER lexical function, 17–17, 17–18
converting data types, 17–17
F$LENGTH lexical function
with F$LOCATE, 17–13
F$LOCATE lexical function
with F$LENGTH, 17–13
F$LOGICAL lexical function, 17–11
F$MODE lexical function
in login procedures, 18–11
F$PARSE lexical function, 17–13, 17–14
F$PID lexical function, 17–7
obtaining process information, 17–7
F$SEARCH lexical function
searching for files, 17–10
using with DELETE command, 17–11
F$STRING lexical function
converting data types, 17–17
F$TRNLNM lexical function, 17–11
translating logical names, 17–11
F$TYPE lexical function
identifying symbols, 17–18
F$VERIFY lexical function, 17–4
Fields, 11–2
See also Sort/Merge utility (SORT/MERGE)
File access
from remote notes, 4–7
using control strings, 4–8
File browsers, 19–12
FILE command
in Mail, 6–13
File names, 4–2
rules for entering, 4–2
specifying a list of, 5–6
using a percent sign (%) wildcard character in, 4–10
using an asterisk (*) wildcard character in, 4–9
valid characters in, 4–2
with null values, 4–10
File protection, 4–16
changing default with F$ENVIRONMENT lexical function, 17–5
File specifications
alternate form for magnetic tapes, 4–11
list of included fields, 4–2
networks, 4–7
node names, 4–6, 4–12
File type field, 4–2
asterisk (*) wildcard character in, 4–9
using a percent sign (%) wildcard character in, 4–10
with null values, 4–10

File types, 4–2
list of defaults, 4–3, 4–4
rules for entering, 4–2
File version numbers
using an asterisk (*) wildcard character in, 4–9
Files
accessing
on a private device, 12–8
on volume sets, 12–8
adding ACEs for security auditing, 19–12
appending to Mail messages, 6–11
applying an alarm to, 19–12
auditing access, 19–10, 19–12
closing from command procedures, 15–17
concatenating, 4–12
controlling the number of versions, 4–5
copying, 4–12
between nodes, 4–12
using Mail, 4–13
copying from remote account, 19–9
creating, 4–11
from Mail messages, 6–10
with COPY command, 4–12
with CREATE command, 4–11
deciding when to use security auditing, 19–12
definition, 4–1
deleting, 4–15
deleting from within command procedures, 15–17
displaying
contents, 4–14, 4–15
using wildcard characters, 4–14
initialization, 6–22
MAIL.MAI, 6–13
mailing
from DCL level, 6–9
using Mail utility, 6–9
modifying, 4–11
naming, 4–1
opening, 13–2
printing, 4–16
printing from a private volume, 12–2
protecting, 4–16, 6–17
protection of confidential, 19–12
protection required for proxy access, 19–9
purging, 4–16
renaming, 4–13
specifying wildcard characters, 4–9
using F$SEARCH lexical function to locate, 17–10
using the /VERSION_LIMIT qualifier, 4–5
version numbers, 4–2, 4–5
Files–11 On-Disk Structure, 12–5
Foreign commands
character limit, 14–3, 14–5
Foreign file specifications
on networks, 4–8
FORTUSER.COM command procedure, C–19
FORWARD command
in Mail, 6–12
Full names, 4–6
Function keys, 3–18
VT200 series terminals, D–1
VT300 series terminals, D–1

G

General identifiers
example, 19–3
Generic device names, 12–10
$GETDVI lexical function, 17–10
GETPARMS.COM command procedure, C–13
sample execution, C–15

H

Hardcopy output
security measures, 2–26
Hardcopy terminals
logout considerations, 2–26
Help
for commands, 2–23
for system messages, 2–23
HELP command, 2–21
in Mail, 6–2
Help Message utility (MSGHLP)
invoking, 2–23
HELP/MESSAGE command, 2–23
Hyphen (-)
in a directory name, 5–9

I

Identifiers
displaying process, 19–2
environmental, 19–2
general, 19–2
UIC, 19–2
IF command
using in command procedures, 15–10
Images
invoking with automatic foreign commands, 14–38
invoking without using symbols, 14–38
redirecting output from, 16–10
Initialization files, 6–22
INITIALIZE command, 12–4
Files–11 On-Disk Structure, 12–5
format, 12–5

Initializing
disk volumes, 12–5
magnetic tape volumes, 12–5
volumes, 12–4
Input
prompting for from command procedures, 16–6
to batch jobs, 18–13
Input files
temporary defaults in a parameter list, 5–6
INQUIRE command
compared to READ command, 16–6
using in command procedures, 15–9
with symbols and batch jobs, 16–7
Insert mode
definition, 3–16
Interactive mode processes, 2–9
Iterations
definition, 15–7

J

Job controllers
affected by shift restrictions, 2–12
Job terminations
imposed by shift restrictions, 2–12
Job trees
definition, 18–4

K

Key definitions
assigning, 3–17
in Mail, 6–22
Key names
in Mail, 6–21
Key sequences, 3–2
Keypads
default in Mail, 6–21
defining in Mail, 6–22
Keys (keyboard)
commonly used Ctrl keys, 3–3
controlling cursor position, 3–19
controlling screen display, 3–20
Ctrl key sequences, 3–2
defining, 3–17
entering DCL commands, 3–18
interrupting DCL commands, 3–18
recalling DCL commands, 3–19
Keywords
in DCL command lines, 3–4

L

Labels
duplicate in command procedures, 15–5
in command procedures, 3–4, 15–4
in local symbol tables, 15–4
Lexical functions
changing process characteristics with, 17–3
converting data types, 17–17
definition, 17–1
evaluating data, 17–17, 17–18
examining character strings, 17–13
extracting parts of character strings, 17–13, 17–14
F$DIRECTORY, 14–23
formatting output strings, 17–15
identifying symbols, 17–18
manipulating character strings with, 17–12
manipulating data types with, 17–17
obtaining information
about files and devices, 17–10
from VMScluster nodes, 17–8, 17–9
process, 17–2, 17–7
queues, 17–7
system, 17–6, 17–7
VMScluster system, 17–6, 17–7
saving process characteristics, 17–4
searching for devices, 17–10
searching for files, 17–10
syntax, 14–21
translating logical names with, 17–11
usage, 14–22
using in command procedures, 14–21
Line editing, 3–16
Linefeed key, 3–19
LISTER.COM command procedure, C–23
sample execution, C–25
Listing files in directories, 5–3
Literal characters
including in command procedures, 15–10
Literal text
character strings, 16–9
Local symbol tables
labels in, 15–4
LOCKPWD flag, 2–7
Log files
contents of, 18–14
examining during execution of batch jobs, 18–15
for batch jobs, 18–14
including command output, 18–15, 18–16
saving, 18–15

Loggin in
to operating system
errors, 2–3
example, 2–2
Logging in
to operating system, 2–2
to remote system, 2–9
Logging out
breaking dialup connection, 2–26
from disconnected processes, 18–9
from remote sessions, 2–24
of the operating system, 2–24
security considerations, 2–25
Logical device names, 12–9
Logical expressions, 14–19, 14–20
Logical name tables, 13–9, 13–10
adding logical names, 13–12
adding to logical name directory, 13–13
contents of process directory table, 13–15 to 13–17
contents of system directory table, 13–17 to 13–20
creating, 13–11
defining protection, 13–13
deleting, 13–22, 13–23
directories, 13–14
displaying, 13–21
displaying directory table structure, 13–22
group, 13–10, 13–11
job, 13–10, 13–11
specifying quotas, 13–14
process, 13–10, 13–11
process-private, 13–10, 13–11, 13–12
shareable, 13–10, 13–12
specifying different tables, 13–9
specifying quotas, 13–14
for job table, 13–14
system, 13–10, 13–11
Logical names, 12–1
access modes, 13–9, 13–10
executive, 13–10
user, 13–10
adding to logical name directory, 13–13
adding to logical name tables, 13–12
behavior
during batch jobs, 13–24
during interactive system operation, 13–23
in nested command procedures, 13–24
when executing a command procedure, 13–23
when opening files, 13–24
compared to symbols, 14–2
concealed, 13–6, 13–7
creating, 13–3
ASSIGN command, 13–3
DEFINE command, 13–3, 13–4
Logical names
creating (cont'd)
for nodes, 13–4 to 13–5
with concealed attribute, 13–6, 13–7
with terminal attribute, 13–6
with translation attributes, 13–6, 13–7
creating a search list, 13–7
using a semicolon, 13–8
with wildcards, 13–8
DCL$PATH, 14–39
defining multiple for one equivalence string, 13–7
definition, 13–1
deleting, 13–22, 13–23
displaying, 13–20
displaying access mode, 13–21
displaying directory table structure, 13–22
displaying logical name tables, 13–21
displaying process permanent files, 13–21
equivalence strings, 13–1
file input, 13–2
file output, 13–2
for nodes
overriding the access control string, 13–5
using in file specification, 13–5
in multiple tables, 13–31
in Phone, 7–3
process-permanent
as file specifications, 13–24
available names, 13–23
redefining, 13–23
SYS$COMMAND, 13–26
SYS$ERROR, 13–26
SYS$INPUT, 13–24
SYS$OUTPUT, 13–25
search lists
search order, 13–30, 13–31
with RUN command, 13–30
with wildcards, 13–30
systemwide, 13–2
terminal, 13–6
translation, 13–26, 13–27
iterative, 13–27, 13–28
modifying, 13–27
search lists, 13–29
system defaults, 13–28
with translation attributes, 13–6, 13–7
Login classes, 2–9
batch, 2–10
dialup, 2–9
interactive, 2–9
local, 2–9
network, 2–10
noninteractive, 2–10
remote, 2–9
restrictions on, 2–11

Login command procedures
definition, 15–45
execution of for batch jobs, 18–11, 18–12
in captive accounts, 15–46
personal, 15–45
executed as batch jobs, 18–14
Login failures
and retries, 2–12
causes of, 2–3, 2–10 to 2–13
messages, 2–8, 19–10
Login messages, 2–7
announcement, 2–7
expired password, 2–3
suppressing, 2–8
Logins
batch, 2–10
changing password during, 2–16
controlling, 2–5
dialup, 2–9
chances to supply password, 2–12
disabled
by break-in evasion, 2–12
for expired accounts, 2–17
interactive, 2–9
local, 2–9
monitoring last, 19–10
network, 2–10
noninteractive, 2–9, 2–10
permitted time periods, 2–12
proxy, 2–10
remote, 2–9
LOGOUT command, 2–24
entering after clearing screen, 2–26
/FULL qualifier, 2–24
/HANGUP qualifier, 2–26
Loops
definition, 15–12
writing, 15–12

M

Magnetic tapes
deallocating drives, 12–3
initializing, 12–5
label format, 12–8
MAIL command, 6–2
/EDIT qualifier, 6–18
/SUBJECT qualifier, 6–10
Mail folders
creating, 6–13
deleting, 6–15
displaying list of, 6–14
MAIL, 6–4
NEWMAIL, 6–3
selecting, 6–14
WASTEBASKET, 6–16
Mail subdirectories
creating, 6–13
Mail utility (MAIL)
appending files to messages, 6–11
deleting messages in, 6–16
distribution lists, 6–7
extracting messages to files, 6–10
initialization files, 6–22
invoking, 6–2
keypad commands, 6–21
reading messages in, 6–3, 6–23
security measures, 6–18
sending files, 6–9
from DCL level, 4–13, 6–9
sending messages over the network, 6–6
setting default editor in, 6–19
summary of commands, 6–23
using EVE in, 6–18, 6–19
using text editors in, 6–18
MAIL$INIT file, 6–22
MAIL$KEYDEF.INI file, 6–22
sample, 6–22
MAILEDIT.COM command procedure, 6–20, C–17
example, C–18
MERGE command, 11–14
See also Sort/Merge utility (SORT/MERGE)
/KEY qualifier, 11–5, 11–15, 11–16
Message count
correcting in Mail with READ/NEW, 6–16
Messages
announcement, 2–7
conventions used in system display, 2–20
copying in Mail, 6–13
during login, 2–7
indicating command line error, 2–20
informational, 2–20
login failures, 2–8
suppressing, 2–8
system responses to commands, 2–19
MFDs (master file directories), 5–1
MOUNT command, 12–6, 12–10
format, 12–6
Mount requests, 12–6
Mounting tapes, 12–7, 12–8
Mounting volumes
and security audit, 19–11
foreign, 12–7
magnetic tape, 12–7, 12–8
MOVE command
in Mail, 6–13
Multiple file specifications
in a parameter list, 5–6

N

Network access control strings, 19–7
Network file specifications
  conventional format, 4–8
  foreign file format, 4–8
  task specification strings, 4–8
  ULTRIX restrictions, 4–8
Network nodes
  local, 4–7
  remote, 4–7
  remote p, 4–7
Networks
  logout, 2–24
  losing connection to remote system, 2–25
  sending mail over, 6–6
Node names
  creating logical names for, 13–4 to 13–5
    overriding the access control string, 13–5
  creating logical names for in file specification, 13–5
  definition, 4–2
  format in file specifications, 4–6
  full, 4–6
  rules for entering, 4–6
Null characters
  in DCL commands, 3–7
Null values
  for file names, 4–10
  for file types, 4–10
Numbers
  comparing, 14–17
  converting to string values, 14–26
  converting using lexical functions, 17–17
  evaluation of, 14–24
  integer values recognized by DCL, 14–14
  internal storage, 14–15
Numeric expressions, 14–15
  and integer operands, 14–15
Numeric overlays, 14–18

O

Object ownership
  changing, 19–3
Objects
  changing security profile, 19–3
  displaying security profile, 19–3
  security profiles, 19–3
Octal numbers
  in a UIC directory specification, 5–10
ON command
  setting Ctrl/Y action routines, 15–36
ON CONTROL_Y command
  enabling Ctrl/Y, 15–40
OpenVMS screen management software
  command recall, 3–15
Operands
  in character string expressions, 14–10
  integer, 14–15
Operators (mathematical)
  order of evaluation, 14–23
Output
  from command procedures, 16–9
  redirecting from commands and images, 16–10
Output strings
  formatting with F$FAO lexical function, 17–15
Overstrike mode, 3–16

P

Parameter lists
  defaults for multiple file specifications, 5–6
  multiple file specifications, 5–6
Parameter qualifiers, 3–9
Parameters
  in DCL command lines, 3–3, 3–4
  passing data to batch jobs with, 16–5
  passing data to nested command procedures with, 16–5
  specifying
    as character strings, 16–3
    as integers, 16–3
    as null values, 16–4
    as symbols, 16–4
  using to pass data to command procedures, 16–3
Password protection, 2–17
  dialup retries, 2–12
Passwords
  acceptable, 2–4
  automatically generated, 2–14
  avoiding password theft programs, 2–18
  changing, 2–3, 2–13
    at login, 2–4
    during login, 2–16
    expired, 2–16
    frequency guidelines, 2–4, 2–18
    secondary, 2–15
    using /NEW_PASSWORD qualifier, 2–16
  dual, 2–5
  expiration, 2–3, 2–16
  failure to change, 2–17
  first, 2–4
  generated, 2–13, 2–14, 2–15
  guessing, 2–4
  guidelines for choosing, 2–3
  high-risk, 2–3

Passwords (cont'd)
incorrect, 2–8
initial, 2–4
length, 2–3, 2–4, 2–13
locked, 2–7
new, 2–16
omission in proxy login, 19–7
open accounts and, 2–7
primary, 2–5, 2–6
protecting, 2–17
reason for changing, 19–10
restrictions, 2–4
retries, 2–12
reuse, 2–3, 2–4
secondary, 2–5, 2–17
entering, 2–6
secure, 2–3
setting a new one, 2–13
supplying during dialups, 2–12
system, 2–5
entering, 2–5
user, 2–4
verifying change of, 2–13
Percent sign (%)
as switchhook character in Phone, 7–3
as wildcard character, 4–10
Personal login command procedures, 15–45
in captive accounts, 15–46
Phone utility (PHONE)
commands, 7–4
qualifiers, 7–3
switchhook character (%) in, 7–3
Physical device names, 12–8
Physical devices
identifying, 12–8
PID numbers, C–12
and process context, 18–2
obtaining using F$PID lexical function, 17–7
Positional qualifiers
definition, 3–9
PRINT command, 4–16 to 4–19
in Mail, 6–17
Print jobs, 4–16
delaying, 4–19
executing, 4–17
list of DCL commands to use with, 4–19
obtaining multiple copies of, 4–19
priorities, 4–17
queue information, 4–17
Print queues
controlling, 4–19
Printing
from a private volume, 12–2
landscape, 4–19
Privileges
VOLPRO, 12–5
Process characteristics
commonly changed, 15–18
Process contexts
list of characteristics, 18–1, 18–2
Processes
and job trees, 18–4
changing characteristics using lexical functions, 17–3
checking status with Ctrl/T, 2–21
connecting to, 2–7, 18–8
creating, 18–1
definition, 18–1
detached, 18–4
disconnected, 2–7, 18–8
logging out, 18–9
displaying process rights identifiers, 19–2
interactive mode, 2–9
reconnecting, 18–8
saving characteristics, 17–4
Program stubs
in command procedures, 15–10
replacing with commands, 15–19
Programs
including in data files, 16–8
Prompts, 3–5
DCL, 2–2
in Mail, 6–2
switchhook in Phone, 7–3
Protection
default, 19–5
directory, 5–7
objects, 19–3
of files, 4–16
in Mail, 6–17
Protection codes, 19–4
access types, 19–5
categories of, 19–4
Proxy accounts, 19–8
default, 19–9
for multiple users, 19–9
for single user, 19–9
general access, 19–9
maximum number allowed, 19–8
naming, 19–9
selecting from multiple, 19–9
Proxy logins, 2–10, 19–7
key characteristic, 19–9
security benefits, 19–7
PURGE command, 4–16
in Mail, 6–16

Q

Qualifiers
command, 3–9
conflicting, 3–10
date formats, 3–11
in DCL command lines, 3–3

Qualifiers (cont'd)
parameter, 3–9
positional, 3–9
specifying values, 3–10
time formats, 3–11
Quotation marks (" ")
in character strings, 16–10

R

READ command
compared to INQUIRE command, 16–6
in Mail, 6–3
/NEW qualifier in Mail, 6–3, 6–16
reading data into command procedures with, 16–6
Reading messages
in Mail, 6–23
Recall buffers, 19–7
erasing, 3–15
RECALL command, 3–14
/ERASE qualifier, 19–7
Recalling commands, 3–14
using arrow keys, 3–14
using RECALL command, 3–15
Record oriented devices
used as output to file specifications, 12–9
Record sorting, 11–2
Records
See also Sort/Merge utility (SORT/MERGE)
entering records from a terminal, 11–16
sorting, 11–9
fields, 11–2
Redirecting output, 12–9
Reinitializing volumes, 12–11
REMINDER.COM command procedure, C–5
sample execution, C–8
Remote nodes
obtaining information about using F$CONTEXT, 17–8, 17–9
Remote sessions
aborting, 2–24
RENAME command, 4–13
REPLY command
in Mail, 6–12
$RESTART symbol
using in command procedures, 15–25
Return key, 3–18
Right arrow key
moving cursor with, 3–19
RMS
file tags in Mail, 6–9
RSX systems
specifying UIC format directory names, 5–10
RUN command
used with search lists, 13–30
with processes, 18–4

S

SEARCH command
in Mail, 6–4
Search lists, 13–1
creating with logical names, 13–7
with wildcards, 13–8
how translated
with wildcards, 13–30
search order, 13–30, 13–31
translation, 13–29
using a semicolon, 13–8
with RUN command, 13–30
Secondary passwords
length, 2–6
Security
administrator, 2–5
audit log files, 19–11
disposing of hardcopy output, 2–26
for devices, 12–2
high-level, 19–10
Security alarms, 19–10
Security audit log files, 19–11
Security auditing, 19–10
account and file access, 19–10
adding ACEs to files, 19–12
deciding when to use, 19–12
messages, 19–11
Security features
account duration, 2–16, 2–17
break-in evasion, 2–12
dialup retries, 2–12
login class restrictions, 2–11
password expiration, 2–16
secure terminal servers, 2–18
security alarms, 19–10
shift restrictions, 2–12
Security profiles
changing objects, 19–3
displaying objects, 19–3
displaying processes, 19–2
displaying users, 19–2
objects, 19–3
processes
displaying, 19–2
users
displaying, 19–2
Security restrictions
login class, 2–11
shift, 2–12
time-of-day, 2–12
Security-auditing events, 19–10
SELECT command
in Mail, 6–4, 6–14
SEND command
in Mail, 6–5, 6–9
/EDIT qualifier, 6–19

Sending files
in Mail, 6–9
Servers
secure terminals, 2–18
SET CONTROL=T command, 2–21
SET DEFAULT command
setting a default device with, 5–6
setting a default directory with, 5–5
SET EDITOR command
in Mail, 6–19
SET ENTRY command, 18–16
qualifiers to, 18–16
SET FOLDER command, 6–14
SET HOST command, 2–9
SET NOCONTROL=Y command
disabling Ctrl/Y, 15–40
SET NOON command
in command procedures, 15–34
with command levels, 15–34
SET ON command
with command levels, 15–34
SET OUTPUT_RATE command, 18–14
SET PASSWORD command, 2–13
automatic password generation, 2–14
/GENERATE qualifier, 2–13
/SECONDARY qualifier, 2–15
SET PREFIX command, 18–14
SET PROCESS command
/NAME qualifier, 18–3
SET PROTECTION command
/DEFAULT qualifier, 19–5
SET QUEUE command
/ENTRY qualifier, 18–16
in Mail, 6–17
qualifiers to, 18–16
SET SECURITY command
/ACL qualifier, 19–6
changing object security profile, 19–3
/PROTECTION qualifier, 19–3, 19–6
SET SIGNATURE_FILE command
in Mail, 6–11
SET SYMBOL command, 14–28
verb string translation, 14–28
SET TERMINAL command, 3–16
/APPLICATION_KEYPAD qualifier, 3–17
/BROADCAST qualifier, 2–21
/INSERT qualifier, 3–16
/LINE_EDIT qualifier, 3–16
/NONUMERIC qualifier, 3–17
/OVERSTRIKE qualifier, 3–16
/WRAP qualifier, 3–17
SET VERIFY command, 18–14
Set-Up key, 2–25
Shift restrictions, 2–12
SHOW DEFAULT command, 5–6
SHOW DEVICES command, 12–2
SHOW ENTRY command, 4–17, 18–10, 18–18
SHOW LOGICAL command, 13–20, 13–21
displaying access mode, 13–21
displaying directory table structure, 13–22
displaying logical name tables, 13–21
displaying process permanent files, 13–21
SHOW PROCESS command, 19–2
/ALL qualifier, 18–1
SHOW QUEUE command, 4–17, 18–18
/FULL qualifier, 18–13
SHOW SECURITY command
displaying security profiles of objects, 19–3
SHOW SYMBOL command, 14–6
SHOW TERMINAL command, 3–16
Signature files
appending to Mail messages, 6–11
SORT command, 11–2
See also Sort/Merge utility (SORT/MERGE)
in command procedure, 11–13
/KEY qualifier, 11–5, 11–6
/NODUPLICATES qualifier, 11–7
/STABLE qualifier, 11–7
using the default key, 11–2
Sort/Merge utility (high-performance), 11–1
Sort/Merge utility (SORT/MERGE)
collating sequence, 11–10
ASCII, 11–11
default, 11–11
EBCDIC, 11–11
multinational, 11–11
NCS, 11–11
user-defined, 11–12
entering records from a terminal, 11–16
improving performance, 11–23
during sorting, 11–24, 11–25
managing work files, 11–25
modifying working set extent, 11–26
/STATISTICS qualifier, 11–23
merging files, 11–14
when sorted by key field, 11–15
merging records
with identical key fields, 11–16
output files, 11–8
qualifiers, 11–26 to 11–45
for input file specification, 11–31
for output file specification, 11–32
in specification files, 11–35
with SORT and MERGE commands, 11–26
running as a batch job, 11–13
command procedure, 11–13
including input records, 11–14
SORT command, 11–2
sorting
noncharacter data files, 11–8
records, 11–9

Sort/Merge utility (SORT/MERGE)
sorting (cont'd)
with identical key fields, 11–7, 11–8
with multiple key field, 11–6
sorting files, 11–2
specification files, 11–17
format, 11–17
including comments, 11–18
order of qualifiers, 11–18
overriding commands, 11–18
using the default key, 11–2
SPAWN command, 18–4
/NOLOGICAL_NAMES qualifier, 18–8
/NOSYMBOL qualifier, 18–8
/NOWAIT qualifier, 18–5
STOP command
in command procedures, 15–14
String assignments
including symbols, 14–7
Subdirectories
creating, 5–3
listing, 5–3
setting default to another, 5–5
syntax, 5–3
SUBMIT command, 18–10
/LOG qualifier, 18–14
passing parameters to command procedures with, 16–5
qualifiers, 18–17
/RESTART qualifier, 18–19, 18–20
specifying multiple command procedures with, 18–12
using with command procedures, 15–24
SUBMIT/AFTER command, 18–11
Subprocesses
and job trees, 18–4
characteristics inherited from parent process, 18–7
context, 18–7
creating, 18–4, 18–5
multiple, 18–5
definition, 18–4
excluding characteristics from parent process, 18–7
exiting, 18–6
transferring control, 18–8
Switchhook character (%)
in Phone, 7–3
SYLOGIN.COM command procedure, 18–10
Symbol scope
definition, 14–28
global, 14–29
local, 14–28
Symbol substitutions
forcing in character strings, 16–10
Symbol tables, 14–26
global, 14–26, 14–27
$RESTART symbol, 14–27
$SEVERITY symbol, 14–27
$STATUS symbol, 14–27
local, 14–26
parameters, 14–26
search order, 14–27
Symbols
abbreviating, 14–4
ampersand (&), 14–32 to 14–33
and arithmetic operations, 14–16
and character strings, 14–8
and expressions, 14–8
and numeric expressions, 14–15
apostrophe (′), 14–31
as DCL command, 14–4
as foreign commands, 14–5
asterisk (*), 14–4
character string expressions, 14–9, 14–10
character string operations, 14–10
compared to logical names, 14–2
comparing numbers, 14–17
concatenating, 14–7
creating with assignment statement, 14–3
defining, 14–3
as a symbol, 14–6
as lexical functions, 14–21, 14–22
character strings, 14–9
defining character strings, 14–9
definition, 14–1
deleting, 14–6
displaying values, 14–6
evaluating, 14–24
global, 14–3, 14–4
identifying with F$TYPE lexical function, 17–18
in string assignments, 14–7
indicating numeric values, 14–14, 14–24
local, 14–3
logical data, 14–18, 14–19
numeric overlays, 14–18
substitution, 14–5, 14–29
automatic, 14–29
command input scanning, 14–34
command parsing, 14–34
expression evaluation, 14–34
forced, 14–30
iterative, 14–34
on data lines, 14–34
operators, 14–31
order of, 14–30, 14–31
phases, 14–33
repetitive, 14–34
using an Ampersand (&), 14–32 to 14–33
using an apostrophe (′), 14–31, 14–32

Symbols (cont'd)
types of characters, 14–8
undefined, 14–37
using as variables, 14–8
SYNCHRONIZE command, 18–20, 18–21
Syntax
file specifications on tape volumes, 4–11
for DCL commands, 3–3
node specifications, 4–6
VMScluster device specifications, 12–10
SYS$ERROR logical name
in log files, 18–14
process-permanent files, 18–14
SYS$INOUT logical name
redefining, 16–5
SYS$INPUT logical name, 16–7
defining as a separate file, 16–8
process-permanent files, 18–13
redefining, 16–7
SYS$OUTPUT logical name
in log files, 18–14
process-permanent files, 18–14
SYS$PRINT logical name, 4–17
SYS.COM command procedure, C–11
sample execution, C–13
System failures
disposing of hardcopy output, 2–26
security implications, 2–18
System messages
See also messages
auditing, 19–11
login failure, 19–10
System objects
using logical names with, 13–1
System privileges
protecting files, 4–16
System security
audit log file, 19–11
Systems
controlling use of, 2–5

T

Tab key, 3–19
Tab stop
advancing to using Ctrl/K, 3–19
Tape volumes
file specifications, 4–11
Tapes
ANSI volume file specification format, 4–11
Task specification strings
on networks, 4–8
Temporary defaults in input file lists, 5–6
Terminals
breaking dialup connection, 2–26
clearing the screen, 2–25, 19–7
control characters, D–3
Terminals (cont'd)
control keys, D–1
controlling access to, 2–5
dialup login, 2–9
disconnected, 18–8
hard copy
disposing of output, 2–26
logout considerations, 2–25
requiring a system password, 2–11
special keys, D–1, D–3
stopping and starting displays, 3–20
system password, requirement for, 2–5
virtual, 2–8, 18–8, 18–9
VT200 series function keys, D–1
VT300 series function keys, D–1
writing data to, 16–9
Testing
command procedures, 15–14
Text editors
creating files, 4–11
displaying files with, 4–14
modifying files, 4–11
THEN command
using in command procedures, 15–10
Time
specifying absolute and delta date and time combinations, 3–13
specifying absolute date and time, 3–11
specifying delta date and time, 3–12
Time-stamping
and verification settings, 17–5
using SET PREFIX command, 18–14
Timeout periods, 2–2
Top-level directories, 5–1
TYPE command
and wildcard characters, 4–14
displaying files on remote nodes with, 4–14
displaying files with, 4–14
displaying files with in EDT, 9–3
TYPE WHOLE command
in EDT, 9–3

U

UAFs (user authorization files), 2–4
account expiration, 2–17
LOCKPWD flag, 2–7
login class restrictions, 2–11
record of last login, 19–10
UIC directory specifications, 5–10
translating to named format, 5–10
UIC identifiers
example, 19–2
UICs (user identification codes)
default protection, 19–5

Unit number field, 12–10
  default value, 12–10
Up arrow key
  recalling commands with, 3–14, 3–19
Users
  displaying process rights identifiers, 19–2

V

Values
  in DCL command lines, 3–3, 3–4
Variables
  assigning in command procedures, 15–9
  assigning using INQUIRE command, 15–9
  definition, 15–7
  determining in command procedures, 15–8
Verification
  enabling during execution of command procedures, 15–16
Viewports
  in Phone, 7–2
Virtual terminals
  disabling, 2–8
  disconnected processes and, 18–8, 18–9
  managing disconnected processes, 18–9
  reconnecting to disconnected processes, 18–8
  restrictions, 18–9
VMScluster systems
  device names, 12–10
    allocation class fields, 12–10
    dual-pathed, 12–10
    format, 12–10
VOLPRO (Volume Protection Override), 12–5
Volume sets
  definition, 12–8
  mounting, 12–6
Volumes, 12–2
  dismounting, 12–11
    allocated devices, 12–11
  displaying information, 12–2
  initializing, 12–4
  label format, 12–8
  mounting, 12–6
    foreign, 12–7
    magnetic tape, 12–7, 12–8
  operator assistance, 12–6
  private, 12–2
VT100-series terminals
  clearing screen, 2–25
VT200-series terminals
  clearing screen, 2–25

W

WAIT command
  synchronizing command procedures, 18–20, 18–21
WASTEBASKET folder
  emptying in Mail, 6–16
Wildcard characters
  asterisk (*), 4–9
  ellipsis (...), 5–8, 5–9
  hyphen (-), 5–9
  in DCL command lines, 3–4
  percent sign (%), 4–10
  using with file names, 4–9
WRITE command
  writing data with, 16–9
  writing strings to records, 17–15