RT–11 System Utilities Manual
Part I

Order Number AA–M239D–TC

August 1991

This manual alphabetically describes 16 utilities beginning with BINCOM and ending with LINK.

Revision/Update Information: This manual supersedes the *RT–11 System Utilites Manual*, AA–M239C–TC.

Operating System: RT–11 Version 5.6

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, March 1983
Revised, July 1984
Revised, November 1985
Reprinted, August 1989
Revised, August 1991

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Any software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. No responsibility is assumed for the use or reliability of software or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

© Digital Equipment Corporation 1983, 1984. 1985. 1989, 1991. All rights reserved. Printed in U.S.A.

The Reader's Comments form at the end of this document requests your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: CTS–300, DDCMP, DECnet, DECUS, DECwriter, DIBOL, MASSBUS, MicroPDP–11, Micro/RSX, PDP, Professional, Q-bus, RSTS, RSX, RT–11, RTEM–11, UNIBUS, VMS, VT, and the DIGITAL logo.

S1481

This document was prepared using VAX DOCUMENT, Version 1.2.

Contents

Preface xi

Chapter 1 RT–11 System Utilities

Definition of a Utility 1–1
Running Utility Programs 1–1
Unsupported Utilities 1–1
Summary of Utilities in Part I 1–2
Summary of Utilities in Part II 1–4
Types of Utilities 1–6
Utilities that Run with DCL Commands 1–6
Utilities that Run as System Jobs 1–6
Utilities that Debug and Alter Programs 1–7
The Command-String Interpreter (CSI) Language 1–8
The Concise Command Language (CCL) 1–10
Prompting Characters 1–11

Chapter 2 Binary-File Comparison Utility (BINCOM)

Binary Comparisons 2–1
Command-Line Syntax 2–2
Using Wildcards with BINCOM 2–3
BINCOM Options 2–4
BINCOM Output-File Format 2–5
Creating a SIPP Command File 2–7
DCL Equivalents of BINCOM Utility Operations 2–8

Chapter 3 Backup Utility (BUP)

Features 3–1
Calling and Terminating BUP 3–2
Command-Line Syntax 3–3
BUP Options 3–5
Summary of BUP Operations 3–7
Backing Up Data 3–9
The Steps of the Backup Operation 3–10
Initializing Backup Volumes (/Z) 3–11
Verifying a Data Transfer (/V[:ONL]) 3–13
Backing Up File(s) 3–15

Backing Up Volumes (/I) 3–16
Backing Up Logical Disks 3–17
Backing Up Files into Logical Disks (subsets) (/R) 3–20
Backing Up to Magtapes 3–21
Listing Directories of Backed-Up Data 3–22
Listing a Directory of Savesets on a Volume (/L) 3–23
Format of a Saveset Directory (a Listing of Savesets) 3–24
Listing a Directory of Files in a Saveset (/S/L) 3–26
Listing the Files in a Logical Disk (/R/L) 3–27
Restoring Backed-Up Data (/X) 3–28
Restoring Complete Savesets 3–29
Restoring Individual Files from Savesets 3–31
Restoring Logical Disks 3–32
Extracting File(s) from a Logical Disk (Subset) /R/X 3–34
Restoring Files Backed Up Prior to RT–11 Version 5.5 3–35
DCL Equivalents of BUP Utility Operations 3–36

Chapter 4 CONFIG, CONSOL, and DATIME Utilities

Configuration Utility (CONFIG) 4–1
Command-Line Syntax 4–1
Options 4–2
Examples 4–4
Console Utility (CONSOL) 4–5
Datime Utility (DATIME) 4–6

Chapter 5 Directory Utility (DIR)

Command-Line Syntax 5–2
DIR Option Descriptions 5–3
DIR Option Summary 5–11
DCL Equivalents of DIR Utility Operations 5–13

Chapter 6 Dump Utility (DUMP)

Command-Line Syntax 6–2
DUMP Options 6–3
Operations With Magtape 6–4
How to Interpret a DUMP Listing 6–5
How to Interpret a DUMP of a Directory 6–7
Example Commands and Listings 6–8
DCL Equivalents of DUMP Utility Operations 6–12

Chapter 7 Device Utility (DUP)

Command-Line Syntax 7–2
Two Types of DUP Options 7–2
DUP Option Summary 7–3
DUP Option Descriptions 7–5
Create Option (/C[/G:value]) 7–6
File Option (/F) 7–8
Image-Mode Copy Option (/I) 7–10
Bad-Block Scan Option (/K) 7–14
Boot Option (/O) 7–16
Squeeze Option (/S) 7–18
Extend Option (/T:value) 7–20
Bootstrap-Copy Option (/U[:dev]) 7–21
Volume-ID Option (/V[:ONL]) 7–23
Wait-for-Volume Option (/W) 7–24
No-Query Option (/Y) 7–25
Directory-Initialization Option (/Z[:value]) 7–26
Changing Default Directory Size (/Z/N:value) 7–28
Changing Volume ID and/or Owner Name (/Z/V) 7–30
Replacing Bad Blocks (/Z/R[:RET]) 7–31
Covering Bad Blocks (/Z/B[:RET]) 7–33
Restoring a Disk (/Z/D) 7–34
DCL Equivalents of DUP Utility Options 7–35

Chapter 8 Single-Line Text Editor (EDIT)

Running EDIT 8–2
Memory Usage 8–3
Two Modes of Operating EDIT: Command and Text 8–4
Special EDIT Key Commands 8–5
Command-Line Syntax 8–7
Character- and Line-Oriented Commands 8–8
Repeating Commands or Command Strings 8–10
Summary of Rules for Entering Commands 8–12
EDIT Command Types and Descriptions 8–14
File Open and Close Commands 8–15
File Input and Output Commands 8–19
Pointer-Relocation Commands 8–24
Search Commands 8–26
Text-Listing Commands 8–29
Text-Modification Commands 8–31
Utility Commands 8–36
EDIT Commands Summary 8–41
EDIT Error Conditions 8–43

Example Editing Session 8–44

Chapter 9 The Error-Logging Package

Forms of the Error Logger 9–1
Generating a System with Error Logging 9–1
Error-Logging 9–2
Functions 9–2
Components 9–4
Descriptions of the Error-Logging Programs 9–6
EL.SYS or ELX.SYS 9–6
ERRLOG.REL 9–6
ELINIT 9–6
ERROUT 9–7
Diagrams of How Error Logging Works 9–8
Using the Error Logger 9–9
With a Single-Job Monitor 9–9
With a Multi-Job Monitor 9–11
Running ELINIT to Enable the Error Logger 9–12
Error-Log Reports 9–13
Displaying, Printing, or Saving Error-Log Reports 9–13
ERROUT Options for Displaying Error-Log Reports 9–14
DCL Equivalents of ERROUT Operations 9–15
Non-MSCP Error-Log Reports 9–16
Storage-Device Report 9–16
Memory Report 9–18
Summary Report 9–20
(T)MSCP Error-Log Reports 9–23
Example of a DU MSCP Report 9–24
Analyzing the Example DU MSCP Report 9–25

Chapter 10 File-Exchange Utility (FILEX)

Operating Systems and File Formats 10–2
FILEX Option Types 10–3
FILEX Option Summary 10–4
FILEX Option Descriptions 10–6
Deleting Files (/D) 10–6
Listing Directories (/L) 10–7
Transferring Between RT–11 and DOS/BATCH or RSTS (/S) 10–8
Transferring to RT–11 from DECsystem–10 (/T) 10–10
Transferring Between RT–11 and Interchange Diskette (/U) 10–11
Writing an Interchange Volume ID (/V[:ONL]) 10–14
Starting and Then Pausing a Transfer (/W) 10–15
Initializing Directories (/Z) 10–16

DCL Equivalents of FILEX Utility Operations 10–17

Chapter 11 Volume Formatting Utility (FORMAT)

Formatting Volumes 11–1
Disks and Diskettes 11–1
Extended Device Units 11–2
Devices at Nonstandard Addresses 11–2
Calling and Terminating FORMAT 11–2
FORMAT Command-Line Syntax 11–3
FORMAT Confirmation Prompts and Messages 11–4
FORMAT Option Descriptions 11–5
FORMAT Option Summary 11–8
DCL Equivalents of FORMAT Operations 11–9

Chapter 12 Logical-Disk Subsetting Utility (LD)

Command-Line Syntax 12–2
LD Option Summary and DCL Equivalents 12–2
LD Option Descriptions 12–3

Chapter 13 The LET Substitution Utility (LET)

Enabling the LET Utility 13–1
Defining Symbols for Substitution 13–1
Deleting LET Substitutions 13–2
Defining Function Keys for Substitution 13–2
LET Options 13–2
Using LET in Your STRTxx.COM File 13–3

Chapter 14 The Librarian Utility (LIBR)

Calling and Terminating LIBR 14–2
Command-Line Syntax 14–2
Linker Object Libraries 14–4
Creating an Object Library File 14–4
Merging Library Files 14–5
Listing the Directory of a Library File 14–6
LIBR Object-Library Option Descriptions 14–7
Combining LIBR Options 14–12
Assembler Macro Libraries 14–13
Creating Macro Libraries 14–13
Options for Creating Macro Libraries 14–13
LIBR Macro-Library Option Descriptions 14–14
LIBR Option Summary 14–15
DCL Equivalents of LIBR Operations 14–16

Chapter 15 The Linker Utility (LINK)

Functions of the Linker 15–1
Calling and Terminating the Linker 15–2
Command-Line Syntax 15–3
Linker Input 15–6
    Input Object Modules 15–6
    Input Library Modules 15–6
Linker Output 15–11
    Output Load Module 15–11
    Output Load Map 15–13
How the Linker Structures the Load Module 15–15
    Assigning Absolute Addresses 15–15
    Creating an Absolute Section 15–15
    Allocating Memory for Program Sections 15–15
Communicating Between Modules (Global Symbols) 15–20
LINK Option Descriptions 15–21
LINK Option Summary 15–42
DCL Equivalents of LINK Utility Operations 15–45

Appendix A The Linker Overlays

Low-Memory Overlays A–2
Low-Memory Overlay Handler (OHANDL) A–10
Extended-Memory Overlays A–14
Extended-Memory Overlay Load Map A–19
Extended-Memory Overlay Handler (VHANDL) A–22
Low- and Extended-Memory Overlays A–28
One Virtual Overlay Segment A–30
Pseudo Overlay Handler (XHANDL) A–34
Separate I and D Space Overlays A–39
I and D Space Overlay Handler (ZHANDL) A–41

Index

Figures

3–1 Summary of the Backup Utility's Backup and Restore Operations 3–8
3–2 Savesets Containing Logical Disks Backed Up as Files 3–17
3–3 Saveset Containing a Logical Disk Backed Up as a Device Image 3–18
15–1 Library Searches 15–8
15–2 Sample Load Map 15–13
15–3 Global Data Section with CON Attribute 15–24
15–4 Global Data Section with OVR Attribute 15–24
15–5 Virtual and Physical Address Space with One Virtual Region 15–35

15–6 Virtual and Physical Address Space with Two Overlay Regions 15–37
15–7 Extended-Memory Partitions that Contain Sharing Segments 15–38
A–1 Sample Overlay Structure for a FORTRAN Program A–3
A–2 Overlay Scheme A–4
A–3 Sample Subroutine Calls and Return Paths A–5
A–4 Memory Diagram Showing BASIC Link with Overlay Regions A–9
A–5 Program Virtual Address Space A–15
A–6 Physical Address Space for Program with Low-Memory Overlays A–16
A–7 Virtual and Physical Address Space A–17
A–8 Memory Diagram Showing Low-Memory and Extended-Memory Overlays A–29
A–9 Memory Diagram Showing Low-Memory I and D Space Overlays A–40

Tables

2–1 BINCOM Command-Line Defaults 2–2
2–2 BINCOM Options 2–4
2–3 DCL Equivalents of BINCOM Utility Operations 2–8
3–1 BUP Options 3–5
3–2 Valid BUP Option Combinations 3–7
3–3 DCL Equivalents of BUP Utility Operations 3–36
4–1 CONFIG Options 4–2
5–1 DIR Options 5–11
5–2 DCL Equivalents of DIR Utility Operations 5–13
6–1 DUMP Options 6–3
6–2 DCL Equivalents of DUMP Utility Operations 6–12
7–1 DUP Option Combinations 7–2
7–2 Summary Descriptions of DUP Options 7–3
7–3 Default Directory Sizes 7–28
7–4 Algorithm for Determining Number of Directory Segments 7–28
7–5 DCL Equivalents of DUP Utility Options 7–35
9–1 Forms of the Error Logger 9–1
9–2 Error-Logging Components for Single-Job Monitors 9–4
9–3 Error-Logging Components for Multi-Job Monitors 9–5
9–4 ERROUT Options 9–14
9–5 DCL Equivalents of ERROUT Operations 9–15
10–1 Supported FILEX Devices 10–1
10–2 FILEX Options 10–4
10–3 DCL Equivalents of FILEX Utility Operations 10–17
11–1 FORMAT Option Summary 11–8
11–2 DCL Equivalents of FORMAT Utility Operations 11–9
12–1 LD Option Summary 12–2
12–2 DCL Equivalents of the LD Utility Operations 12–2
14–1 LIBR Macro Options 14–13
14–2 LIBR Option Summary 14–15
14–3 DCL Equivalents of LIBR Utility Operations 14–16

15–1 Linker Prompting Sequence 15–4
15–2 Absolute Block Parameters 15–11
15–3 PSECT Attributes 15–16
15–4 Section Attributes 15–18
15–5 PSECT Order 15–19
15–6 Linker Options 15–42
15–7 DCL Equivalents of LINK Utility Operations 15–45

Preface

Audience

This manual is written for experienced users of the RT–11 operating system.

Document Structure

This manual alphabetically describes the following utilities:

BINCOM
BUP
CONFIG
CONSOL
DATIME
DIR
DUMP
DUP
EDIT
ERROR LOGGER (EL, ELINIT, ERRLOG, and ERROUT)
FILEX
FORMAT
LD
LET
LIBR
LINK

See Appendix A of Part II of this manual for a description of BATCH.

Conventions

The following conventions are used in this guide.

Convention	Meaning
Braces ({ })	In command syntax examples, braces enclose options that are mutually exclusive. You can choose only one option from the group of options that appear in braces.
Brackets ([ ])	Square brackets in a format line represent optional parameters, qualifiers, or values, unless otherwise specified.
lowercase characters	In command syntax examples, lowercase characters represent elements of a command for which you supply a value. For example: `.ASSIGN dev: WF` RET
UPPERCASE characters	In command syntax examples, uppercase characters represent elements of a command that should be entered exactly as given.
number. $number_{10}$	A number followed by a period or the subscript ten indicates a decimal number.
number $number_8$	A number without a decimal point (period) or followed by the subscript eight is an octal number, unless otherwise indicated. For example, 128. or 128_{10} is 128 (decimal) and 126 or 126_8 is 126 (octal).
RET	RET in examples represents the RETURN key. Unless the manual indicates otherwise, terminate all commands or command strings by pressing RET.
CTRL/*x*	CTRL/*x* indicates a control key sequence. While pressing the key labeled Ctrl, press another key. For example: CTRL/C.
Italics	*Italics* indicate a book title, information and error messages quoted in paragraphs, syntax elements of a command line when referenced in a paragraph, or, occasionally, a word highlighted in a paragraph because of its importance.

Associated Documents

Basic Books

- *Introduction to RT–11*
- *Guide to RT–11 Documentation*
- *RT–11 Commands Manual*
- *PDP–11 Keypad Editor User's Guide*
- *PDP–11 Keypad Editor Reference Card*
- *RT–11 Quick Reference Manual*
- *RT–11 Master Index*
- *RT–11 System Message Manual*
- *RT–11 System Release Notes*

Installation Specific Books

- *RT–11 Automatic Installation Guide*
- *RT–11 Installation Guide*
- *RT–11 System Generation Guide*

Programmer Oriented Books

- *RT–11 IND Control Files Manual*
- *RT–11 System Macro Library Manual*
- *RT–11 System Subroutine Library Manual*
- *RT–11 System Internals Manual*
- *RT–11 Device Handlers Manual*
- *RT–11 Volume and File Formats Manual*
- *DBG–11 Symbolic Debugger User's Guide*

Chapter 1
RT–11 System Utilities

Definition of a Utility

A utility is a program (or, in a few cases, a group of programs called a package, with a master program controlling the whole) that provides you with a set of related general-purpose functions, such as editing, linking, or backing up files. RT–11 distributes many utility programs. Most of these programs are described in this manual, though a few, such as HELP, INDEX, and SL (the Single-Line command editor) are described in the *Introduction to RT–11*.

Running Utility Programs

Most of the utility chapters describe how to run a utility program with the R command. See Part I of the *Introduction to RT–11* for an introduction to the RT–11 utility programs and the various ways you can run them.

Unsupported Utilities

An unsupported utility is a utility distributed by Digital, but not guaranteed to be compatible with future releases of that utility and not guaranteed to be distributed in future releases of RT–11. However, SPRs (Software Performance Reports) are answered for all programs distributed with RT–11.

The following utilities described in this manual are unsupported in the sense just explained:

- CONFIG
- CONSOL
- DATIME
- LET
- NITEST
- RTMON
- SPLIT
- TRANSFER
- VBGEXE

Summary of Utilities in Part I

The *RT–11 Utilities Manual, Part I*, alphabetically describes the following utilities:

BINCOM

Compares two volumes or binary files and lists the differences between them.

BUP

Backs up and restores RT–11 files or volumes.

CONFIG

Enables you to determine:

- Whether a specified handler is installed.
- Whether a specified memory location exists in a system.
- Whether the contents of a specified location match a specific value.
- Whether an MSCP device unit contains fixed or removable media and whether that media is available.

CONSOL

Changes the system console on systems that do not include multiterminal support. The terminal interface between the two terminals must be DL.

DATIME

Ensures the entry of the current date and time.

DIR

Lists a wide range of directory information.

DUMP

Translates the binary data in all or part of a file or volume into formatted octal words and/or bytes, ASCII characters, and/or Radix–50 characters

DUP

Maintains file-structured devices.

EDIT

Enables you to edit single lines of text files on hardcopy terminals.

Error Logger (EL, ELINIT, ERRLOG, and ERROUT)

Monitors the hardware reliability of your computer system.

FILEX

Converts files from one format to another so that you can use them with various operating systems.

FORMAT

Formats disks and diskettes, converts diskettes either to single- or double-density, and formats volumes so they are usable by RT–11.

LD

Enables you to define and access logical disks by associating a logical-disk unit number with a file.

The LD utility functions as a device handler when you load it and as a utility when you run it.

LET

Enables you to substitute symbols for characters and strings in a KMON command line.

LIBR

Enables you to:

- Create, update, modify, list, and maintain object library files.
- Create macro library files for the MACRO–11 assembler.

LINK

Converts object program modules to a format suitable for loading and execution.

Summary of Utilities in Part II

The *RT–11 Utilities Manual, Part II*, describes the following utilities. They are in alphabetical order with one exception, BATCH, which is described in the appendix.

MACRO

Compiles MACRO–11 programs.

NITEST

Lets you verify that communications are possible between one machine on an Ethernet running RT–11 V5.2 or greater and another machine on the same Ethernet.

ODT/VDT

Helps you to debug assembly language programs.

PAT

Enables you to add code to a relocatable binary-object module, without having to perform any octal calculations.

PIP

Enables you to transfer files between any RT–11 supported devices and to merge, rename, delete, and change the protection status of files.

QUEUE

Sends files to any valid RT–11 device.

RESORC

Examines the currently running RT–11 system and displays information about the status of the monitor and the system configuration.

RTMON

Runs as a foreground/system job and provides a real-time display of system activity.

SIPP

Enables you to examine code and to make code modifications to any RT–11 binary-output file that exists on a random-access storage volume.

SLP

Enables you to make code modifications to any RT–11 file that exists on a random-access storage device.

SPLIT

Divides a file along the block boundaries you specify and copies each segment to a separate file.

SPOOL

Automatically intercepts all data directed to the printer or other designated output device, stores it, and then forwards it to the printer or output device, while allowing you to use your terminal.

SRCCOM

Compares two text files and lists the differences between them.

TRANSFER/TRANSF

While running on a host operating system, copies files from an RT–11 stand-alone processor to the host processor or from the host to the stand-alone.

VBGEXE

Creates a pseudo unmapped-monitor environment enabling you to run programs faster and with less low-memory space than the programs would otherwise require.

VTCOM

Enables you to connect your stand-alone RT–11 operating system to another computer's operating system and to communicate between the two operating systems.

BATCH

Allows you to run programs without programmer interaction.

Types of Utilities

RT–11 distributes the following three types of utilities:

- Utilities whose functions you can run with DCL commands
- Utilities you can run as system jobs
- Utilities that enable you to debug and alter programs

Some of these types overlap; for example, you can use DCL commands to run utilities as system jobs. You can categorize utilities in other ways too; for example, some utilities come as a package of programs that work together rather than as one program. These three types, however, are primary ways you can look at the RT–11 utility programs.

Utilities that Run with DCL Commands

Most of the utilities interact with the DCL commands described in the *RT–11 Commands Manual*. You can take advantage of nearly all the capabilities provided by RT–11 utilities by using these commands.

Note, though, that DCL commands call utility programs, which then perform the functions specified by the DCL commands. Some utility functions, however, are not available through DCL commands. For the relationship between a utility program and DCL commands, see the DCL Command and Utility Program Equivalents section in that utility chapter. If there is no such section, then that utility has no CSI commands that are related to DCL commands. For further information on the relationship between DCL commands and utilities, see:

- Part I, Basics for the New User, in the *Introduction to RT–11*.
- Appendix B, DCL Command and Utility Program Equivalents, in Part II of the *RT–11 System Utilities Manual*.

Utilities that Run as System Jobs

There are four utilities that come as packages which you can run as system jobs on a multi-job monitor:

- The error-logging package (EL and ELX, ELINIT, ERRLOG, ERROUT (.REL and .SAV))

  You can run the Error Logger under a single-job monitor as well as under a multi-job monitor, though it is not encouraged. See the error-logging chapter for more information.

- The QUEUE package (QUEUE, QUEMAN, QUFILE)
- The SPOOL package (SPOOL (.REL and .SAV), SP and SPX)
- The VTCOM package (VTCOM (.REL and .SAV), XL and XLX, XC and XCX, TRANSFER/TRANSF (.EXE,.TSK,.SAV))

See the *Introduction to RT–11* for a tutorial of system jobs.

Utilities that Debug and Alter Programs

See the description of the DBG debugging utility in the *DBG–11 Symbolic Debugger User's Guide* for the most powerful RT–11 debugging program. For other debugging utilities, see the descriptions of the following four utility programs in Part II of this manual:

- ODT (On-line Debugging Technique) — Chapter 19
- PAT (Object-Module Patch Program) — Chapter 20
- SIPP (Save-Image Patch Program) — Chapter 25
- SLP (Source-Language Patch Program) — Chapter 26

The Command-String Interpreter (CSI) Language

When you run an interactive utility as a utility (rather than through a DCL command), the Command-String Interpreter (CSI) displays an asterisk (*) at the left margin of the terminal, indicating that it is ready to accept input.

You can use the SL (Single-Line) command editor to edit your input for a CSI command in the same way you can use that editor to edit DCL commands. See the *Introduction to RT–11* for a tutorial and see the *RT–11 Commands Manual* for a summary description.

CSI Command-Line Syntax

The following is the general syntax for entering a CSI command. See the individual utility chapters for the specific syntax required by that utility.

output-filespecs/options=input-filespecs/options

or

dev:filnam.typ[n],...dev:filnam.typ[n][/op[:val]...]=dev:filnam.typ,...dev:filnam.typ[/op[:val]...]

where:

dev: specifies either a logical or physical device name. If you do not supply a device name, RT–11 uses the default device (DK), or whatever device you specify for the first file in a list of input or output files. For example, consider the following command:

```
*DU1:FIRST.OBJ,LP:=TASK.1,DL1:TASK.2,TASK.3
```

This command is interpreted as:

```
*DU1:FIRST.OBJ,LP:=DK:TASK.1,DL1:TASK.2,DL1:TASK.3
```

filnam.typ is the name of a file (consisting of one to six alphanumeric characters followed optionally by a period and a zero- to three-character file type). No spaces or tabs are allowed in the file name or file type. As many as three output and six input files are allowed. If you omit the dot and the file type, RT–11 applies a default file type if the specified program has one.

[n] is an optional declaration of the number of blocks you need for an output file; n is a decimal number (up to 65,535) enclosed in square brackets immediately following the output filnam.typ to which it applies.

/options or **[/op[:val]/...]**

is one or more single-letter options. Some options have a value as an argument; this value can be either an octal number or one to three alphanumeric characters (the first of which must be alphabetic) that the program converts to Radix–50 characters. The default numbering system is octal unless otherwise indicated. To express a decimal number you must place a period after the number; and to express more than one value for those options, you specify each value with a beginning colon; for example:

```
/T:24.:JAN:90.
```

You can use a minus sign (-) to denote negative octal or decimal numbers.

Note that most DCL option values are interpreted as decimal by default, while most CSI option values are interpreted as octal by default. See the individual option descriptions for more information.

You can mix octal, Radix–50, and decimal values.

=

Separates the output and input fields. If there are no output files, you can use the < sign in place of the = sign, and you can omit this separator entirely.

The Concise Command Language (CCL)

The Concise Command Language (CCL) is a shortened form of the CSI command language. It is shortened in the sense that it enables you to run a program and pass it a command string on a single command line.

When you type a CCL command, the keyboard monitor translates the command into a RUN SY: command followed by the program name you specify and one or more lines of file specifications and options for that program. When the operation completes, control returns to the keyboard monitor and it prompts you for another command.

The general syntax for issuing a CCL command is:

.program-name input-file(s)[/option/...] [output-file(s)[/option/...]]

or

.program-name output-file(s)[/option/...]=input-file(s)[/option/...]

where:

program-name	specifies the RT–11 utility program you want to run.
input-file(s)	specifies the device, file names, and file types of the input files. Implicit wildcards are not allowed in file specifications; you must use the wildcard symbols * and %.
output-file(s)	specifies the device, file names, and file types of the output files. Implicit wildcards are not allowed in file specifications; you must use the wildcard symbols * and %. In the first syntax line, output file specifications are optional.
/option(s)	specifies the single-character CSI options for each utility program.

Examples

1. This command copies all files on DU0 with the file type MAC created on or after January 12, 1990 to DU1:

   ```
   .PIP DU0:*.MAC/I:12.:JAN:90. DU1:*.* RET
   ```

2. This command, using the alternate way of formatting, achieves the same results as the preceding command:

   ```
   .PIP DU1:*.*=DU0:*.MAC/I:12.:JAN:90 RET
   ```

3. This command calls KEX to inspect the file PROG1.MAC:

   ```
   .KEX PROG1.MAC/I RET
   ```

Prompting Characters

The following table summarizes the characters RT–11 displays either to indicate that the system is waiting for your response or to specify which job (foreground, system, or background) is producing output.

Character	Description
. (dot)	The keyboard-monitor prompt indicates the keyboard monitor is waiting for a command.
^	The circumflex terminal prompt indicates the terminal is being used as an input file. This prompt requests you to enter information from the keyboard. Pressing CTRL/Z marks the end-of-file. See the *RT–11 Commands Manual* for descriptions of the special function keys you can use with the command.
>	The angle-bracket output prompt indicates which job (foreground, system, or background) is producing the output that currently appears on the terminal: **B>** Indicates output from the background job. **F>** Indicates output from the foregroung job. **jobname>** Indicates output from the system job specified by *jobname*.
*****	The CSI prompt indicates the current system utility program is waiting for a command.

Chapter 2
Binary-File Comparison Utility (BINCOM)

The RT–11 Binary-File Comparison Utility (BINCOM) compares two volumes or binary files and lists the differences between them. BINCOM can either display the results at the terminal or printer, or store them in a file.

With BINCOM, you can:

- Compare similar files in binary code (compiled or assembled files).
- Compare two executable programs by quickly telling whether two data files are identical.
- Verify whether two versions of a program produce identical output files when given identical input files.
- Create a command file that invokes the save-image patch program (SIPP, described in Part II, Chapter 25 of this manual) to patch one version of a file so that it matches another version. The Creating a SIPP Command file section describes the procedure you can use to create a command file for SIPP.

Binary Comparisons

BINCOM examines the two input files word by word (or byte by byte), looking for differences. When BINCOM finds a mismatch, it lists:

- The block number and offset within the block at which the difference occurs.
- The octal values from each input file.
- The logical exclusive OR of the two values. This last number helps you find the bits that are different in the two values.

Calling and Terminating BINCOM

To call BINCOM from the system device, respond to the dot (.) prompt displayed by the keyboard monitor by typing:

```
.R BINCOM RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by only pressing RETURN, BINCOM displays its current version number. You can type CTRL/C to halt BINCOM and return control to the monitor when BINCOM is waiting for input from the terminal. You must type CTRL/C twice to abort BINCOM at any other time. To restart BINCOM, type R BINCOM or REENTER and press RETURN in response to the monitor's dot.

Command-Line Syntax

BINCOM accepts command strings with the following syntax:

[out-spec[/option]][,patch-spec[/option]]=file-1,file-2[/option...]

where:

out-spec	specifies the file or volume to which you want the differences between the two files or volumes you are comparing sent.
patch-spec	specifies the file that you can run as a command file; it will contain the commands necessary to patch file-1 so it matches file-2.
file-1	specifies the first file to be compared.
file-2	specifies the second file to be compared.
option	is one or more of the options listed in Table 2–2.

Table 2–1 lists the command-line defaults.

Table 2–1: BINCOM Command-Line Defaults

Default Device or File Type	Description
Terminal	Output device
DK	Input device
DIF	File type for a differences output file
COM	File type for a SIPP command output file

Note that you must always specify the file type for input files.

Restriction

A BINCOM comparison of magtapes is valid only on those tapes having 512-byte blocks.

Using Wildcards with BINCOM

You can use wildcards to perform multiple binary-file comparisons by typing only one command line. However, you can use wildcards only to compare files; you cannot use wildcards when creating a SIPP indirect command file.

You can use wildcards in either input file specification (file-1 or file-2). A different type of comparison is performed depending on whether you use wildcards in only one or in both of the input file specifications:

- If you use wildcards in only one of the input-file specifications, BINCOM compares the file you specify without any wildcards to all variations of the file specification that contains wildcards.

  The wildcards represent that part of the file specification to be varied. You can use this method to compare one particular file to several other files. For example, when the following command line is executed, BINCOM compares the file TEST1.SAV on device DU0 to all files on device DU1 with the filename TEST2:

  ```
  *TEST=DU0:TEST1.SAV,DU1:TEST2.*  RET
  ```

  You can send the results of all the comparisons to a file on a volume rather than to the console by specifying an output file. In the last example, all differences from the comparisons are sent to the file TEST.DIF on device DK.

- If you use wildcards in both input-file specifications, BINCOM compares pairs of files, that is, each input file specification is compared to only one other input file specification.

  The wildcards represent a part of the file specifications that you want to be the same in both files being compared. You can use this method to compare several pairs of files. For example, when the following command line is executed, BINCOM compares pairs of files; the first input file in each pair has the file name PROG1, and the second has the file name PROG2. The file type of both files in each pair must match:

  ```
  *DU0:PROG1.*,DU1:PROG2.*  RET
  ```

  BINCOM first searches DU0 for a file with the name PROG1, and takes note of its file type. Then, BINCOM searches DU1 for a file with the name PROG2 and the same file type as PROG1. If a match is found, BINCOM compares the two files and lists the differences on the terminal (or sends the differences to an output file, if one is specified). BINCOM then searches DU0 (for more PROG1 files) and DU1 (for PROG2 files with matching file types).

BINCOM Options

Table 2–2 summarizes the options that you can use with BINCOM. Except for the /O option, you can place these options anywhere in the command line, but it is conventional to place them at the end of the command line.

Table 2–2: BINCOM Options

Option	Function
/B	Compares the input files byte by byte. If you do not specify this option, BINCOM compares the files word by word.
/D	Compares two entire volumes starting with block 0. If one volume is longer, BINCOM displays a message and compares the volumes up to the point where the shorter volume ends and the longer one continues. Invalid when creating a SIPP command file.
/E:value	Ends comparison at the block specified by *value*, where *value* is an octal number. If you do not include this option, BINCOM ends the comparison when it reaches end-of-file on one of the input files, or end-of-device on one of the input devices.
/H	Types on the terminal the list of available options.
/O	Creates a differences output file or patch file, even if there are no differences between the two input files. If you enter this option after the SIPP indirect command file, BINCOM creates a SIPP indirect command file whether or not differences exist. You can enter this option at the end of the command line if you want both output files. This option is useful in BATCH streams to prevent later job steps from failing because BINCOM did not create the expected control file.
/Q	Suppresses the display of the differences and displays only the message *?BINCOM-W-Files are different* or *?BINCOM-W-Devices are different* if applicable (or *?BINCOM-I-No differences found*). This option is useful in BATCH control files when you want to test for differences and perhaps abort execution, but do not want the log file filled with output.
/S:value	Starts the comparison at block specified by *value*, where *value* is an octal number.

BINCOM Output-File Format

If you include an output-file specification in the command line, BINCOM creates a file that contains the differences between the two input files or devices. If you do not specify an output file, BINCOM displays the differences only on the terminal. If you include the /Q option, BINCOM does not display the differences and does not create an output file.

The first line of the differences listing is a header line that identifies the files or devices you are comparing. Next, BINCOM displays a blank line and then lists the differences between the two files or devices. Each differences line has the following format:

bbbbbb ooo/ ffffff ssssss xxxxxx

where:

bbbbbb	is the octal number of the block that contains the difference
ooo	is the octal offset within the block
ffffff	is the value in the first file or device
ssssss	is the value in the second file or device
xxxxxx	is the logical exclusive OR of the two values

If there are several differences in a block, BINCOM displays the block number only once for that block. Thus, each time a block number appears, it indicates that the differences being displayed are in a new block.

If you specify the /B option to compare byte by byte, BINCOM displays ffffff, ssssss, and xxxxxx as 3-digit, octal, byte values.

Messages

- When BINCOM reaches the end of one of the input files or devices, it checks its position in the other. If the files or devices have different lengths, BINCOM displays the message:

  ```
  ?BINCOM-W-File is longer DEV:FILNAM.TYP
  ```

  or

  ```
  ?BINCOM-W-Device is longer DEV:
  ```

- BINCOM displays the following message on the terminal if it found any differences:

  ```
  ?BINCOM-W-Files are different
  ```

  or

  ```
  ?BINCOM-W-Devices are different
  ```

- If the two files or devices are identical up to that point, BINCOM displays this message:

```
?BINCOM-I-No differences found
```

If you include a SIPP command-file specification in the command line, BINCOM creates a file that is a valid command file for the save image patch program (see Part II, Chapter 25). This command file contains commands that instruct SIPP to patch the first input file so that it matches the second input file. If you want BINCOM to create only the patch file, enter a comma before the patch-file specification in the command line in place of the output-file specification.

Examples

1. This example compares files TEST1.TST and TEST3.TST, both on device DK. Notice that there are no output files and no options in the command line:

```
.R BINCOM
*TEST1.TST,TEST3.TST
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
?BINCOM-W-Files are different
```

   Notice the fourth line. The third number, 051511, specifies the contents of location 2, block 0, in file TEST1.TST. The fourth number, 051502, specifies the contents of the same location in file TEST3.TST. The last number is the logical exclusive OR of the two values.

2. This example specifies the output file FOO1 as the file in which to store the differences between TEST1.TST and TEST3.TST:

```
.R BINCOM
*FOO1=TEST1.TST,TEST3.TST
?BINCOM-W-Files are different
```

3. The contents of file FOO1 from the preceding example follow. Note that FOO1 has the default file type DIF:

```
.TYPE FOO1.DIF
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
```

Creating a SIPP Command File

You can use BINCOM to create a command file that invokes the save-image patch program (SIPP, described in Part II, Chapter 25) to patch one version of a file you are comparing to match the other version. You specify this indirect file as the second output file in the CSI command string. If you wish to create only the indirect file as output, place a comma before the output-file specification in the command line, in place of the first output-file specification.

The example that follows specifies FOO2 as the patch output file. This file will contain the commands necessary to patch file TEST1.TST so it matches TEST3.TST. Notice the comma before the patch file specification. The comma indicates that a differences output file is not requested, resulting in the displaying of all the differences at the terminal when the command is executed:

```
.R BINCOM RET
*,FOO2=TEST1.TST,TEST3.TST RET

BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
?BINCOM-W-Files are different
```

The contents of file FOO2 follow. Note that BINCOM assigns to this file the COM file type:

```
.TYPE FOO2.COM RET

R SIPP
DK:TEST1.TST/A
000000
000000002
051502
^Y
^C
```

You can run the file FOO2 from the previous example as an indirect command file to make TEST1.TST match TEST3.TST. You can do this with the following command, when typed in response to the keyboard monitor dot:

```
@FOO2.COM
```

DCL Equivalents of BINCOM Utility Operations

Table 2–3 lists the DCL DIFFERENCES/BINARY command options that are equivalent to BINCOM utility operations.

The first part of the table lists that part of the BINCOM command syntax that is equivalent to a DIFFERENCES/BINARY option. The rest of the table alphabetically lists all the BINCOM options having DCL equivalents. Those BINCOM options not having DCL equivalents are not listed.

Table 2–3: DCL Equivalents of BINCOM Utility Operations

BINCOM Syntax/Option	DIFFERENCES/BINARY Option
filespec=	/OUTPUT:filespec
LP:=	/PRINTER
TT:=	/TERMINAL
,SIPP-spec=	/SIPP:filespec
[size]	/ALLOCATE:size
/B	/BYTES
/D	/DEVICE
/E:value	/END[:value]
/O	/ALWAYS
/Q	/QUIET
/S:value	/START[:value]

Chapter 3
Backup Utility (BUP)

The Backup Utility (BUP) is a file-transfer program for storing files, especially large files, volumes, and logical disks. BUP is especially designed to efficiently and speedily back up large amounts of information.

Use the on-line index, INDEX, to find the latest information about topics related to BUP.

See the *Introduction to RT–11* (both the Basics section and the Features section) for a tutorial of:

- How to use BUP's DCL backup commands
- How BUP backup operations compare with those of DUP and PIP
- Type of backup operation best suited for each utility (BUP, DUP, and PIP)

Features

BUP has the following features:

- You can use BUP to make file or device-image backups from disks to any media. Unlike PIP and DUP, BUP is used only to make and restore backups and is more efficient than PIP or DUP in making backups.
- Backs up files or RT–11 volumes of any size. Is the only utility that lets you back up files that are larger than a single volume of the backup media.
- Scans for bad blocks when it initializes a new disk backup volume.
- Uses a more efficient verification procedure than that of PIP or DUP. Verifies backed-up data in a quick separate pass, rather than in single blocks during the backup procedure.
- Places backed-up files or a device image into a saveset on the backup volume, and assigns the saveset a name.

  A saveset can be thought of as a container that holds one or more files or an RT–11 volume or device image from a single backup operation. Each saveset is the result of a single backup operation and is stored in the format of an RT–11 volume.
- Creates more than one saveset on a backup volume, if the volume is large enough. Since individual files are enclosed in a saveset, you do not have to worry about multiple copies of the same file (from different backups) overwriting each other.

- Creates savesets on magtape that are easily transportable to VAX processors running the VMS operating system. Once the files on saveset are transported to a VAX, those files are easily read and manipulated.
- Lists a directory of savesets located on a backup volume(s).
- Lists a directory of the files residing in a saveset.
- Supports wildcards in file specifications.
- Restores whole savesets or individual files in a saveset to their original form on a volume.
- Backs up files to automatically created logical disks that contain exactly sufficient disk space to hold the files.
- Lists directories of logical disks without your having to mount them.
- Restores files from unmounted logical disks.

Calling and Terminating BUP

To call BUP from the system device, respond to the keyboard monitor dot prompt (.) by typing:

```
.R BUP RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to type a command string. If you merely press RETURN at this point, BUP displays its current version number and prompts you again for a command string.

You can press CTRL/C to terminate BUP and return to the monitor when BUP is waiting for input from the terminal. However, you must press CTRL/C twice to terminate BUP at any other time.

NOTE

Before you run BUP under the FB monitor, unload unnecessary foreground jobs to gain more memory. This produces more efficient magtape streaming.

Command-Line Syntax

The following CSI command-line syntax presumes you are at the asterisk prompt, having already issued the command to run the BUP utility:

out-spec,,listing-spec=in-spec[/options]

where:

out-spec specifies the device in which you will mount the output volume(s) for the backup operation, and the saveset (or logical-disk) file name for the backup.

Note:

- You must copy to a saveset even when you are backing up a volume.
- You can type only one output specification for backing up your data.
- You can use wildcards on input files but not on saveset specifications.
- Output volumes for backing up data as savesets must be initialized by BUP. See the Initializing Backup Volumes (/Z) section for information on initializing backup volumes.

Defaults:

- BACKUP is the default output file name for a file being backed up.
- ddn (the name of the device being backed up) is the default output file name for a device-image backup.
- BUP is the default output file type for all saveset backups (file and device-image) except for logical-disk savesets.
- DSK is the default output file type for all logical-disk saveset backups.

out-spec-2 (,) is reserved. You cannot specify out-spec-2, but you must include the comma representing its location in the command line when you specify the listing-spec (the third output specification).

listing-spec is a directory listing of your backed-up data. The listing can be of savesets on a backup volume or of files in a backup saveset or subset (logical disk). By default, a directory listing is displayed on your terminal. To send that listing to the printer, specify LP:.

Note: You cannot specify both an out-spec and a listing-spec. And, unless you use the /L option for specifying the listing-spec, you must type two commas before the listing-spec on the BUP command line. The commas are place holders placing the listing-spec as the third output specification on the command line. For example, the following commands respectively list the directory of the saveset file MYBACK.BUP on the printer and in the file MYBACK.DIR:

```
*,,LP:=DU1:MYBACK.BUP/S RET
```

or

```
*,,DU0:MYBACK.DIR=DU1:MYBACK.BUP/S RET
```

in-spec You can type up to six input specifications. If you specify a saveset or logical-disk subset, then it must be the first input specification and cannot contain wildcards; otherwise, you can use wildcards. DK is the default input device.

/options See the BUP Options section for descriptions of the options you can specify with BUP. If you have no options, BUP assumes you want to back up a file(s); see the Backing Up File(s) section.

BUP displays a list of all the files affected by the operation as the operation progresses, unless you specify the /W (nolog) option.

Input and Output Volumes

You can use random-access volumes and logical disks as either input or output volumes for both backup and restore operations. Magtapes, however, can be used only as output volumes for a backup operation, and only as input volumes for a restore operation.

Bad Blocks on the Input Volume

By default, BUP successfully tolerates up to 25 bad blocks on the input device when backing up a disk to a backup volume or when restoring from a backup volume to a disk (copy back).

BUP issues a warning message each time it encounters a bad block on the input device, then continues to back up or restore. If BUP encounters more than 25 bad blocks on the input device, BUP issues a fatal error message and the operation is stopped.

You can change the number of bad blocks BUP accepts on the input volume by using the customization procedure described in the *RT–11 Installation Guide*.

BUP Options

Table 3–1 lists all the BUP options and their operations. The sections following this table describe all the options in detail with the exception of the /W option. The /W (nolog) option is the only option not described elsewhere.

If you specify none of these options, BUP assumes that you want to back up a file(s).

Table 3–1: BUP Options

Option	Function
/E	Used only with /X. Allows you to restore SYS files to SY when using wildcards. This is to prevent you from accidentally losing SYS files.
/F	Used only with /X. When used with the /X and /I options (/X/I/F), it is equivalent to /X and restores one or more files from a backup volume saveset. When used with /X (/X/F), it restores an entire saveset to one file.
/G	Inhibits the bad-block scan on disk output volumes. Use /G only on disk output volumes that you know contain no bad blocks. The default operation is to scan each disk output volume for bad blocks.
/I	Backs up an entire volume to smaller volumes in image mode. Also used with /X during restore operations.
/L	Displays a directory of a backup volume. The /L/R option displays a directory of a logical-disk file.
/M	Inhibits rewinding magtape before appending next saveset to that magtape. Increases the speed of backup operations but also stops saveset name verification (the magtape must rewind to check for unique saveset names). The default operation is to rewind the magtape before appending the next saveset to that magtape. You must explicitly load the magtape handler before using the /M option. If the handler is fetched, an automatic rewind operation is always performed.
/R	Creates logical-disk images of the files you want to back up. The /R/L option lists a directory of a logical-disk file, and /R/X restores files from a logical disk.
/S	Indicates the saveset containing a file you want to restore or the saveset from which you want to obtain directories.
/V[:ONL]	Verifies that the output data matches the input data in a backup or restore operation. The verification procedure is slightly different, depending on whether you do a verification during a backup, after a backup (before the original data is changed), or after a restore operation. /V verifies data in a backup operation; /V/X verifies data in a restore operation; and /V:ONL/X verifies data in a previous backup operation (without restoring the data). The /V:ONL option is valid only when used with the /X option and when the original data is as it was before a backup. All verification procedures check device (read) errors and check the data integrity of blocks read.

Table 3–1 (Cont.): BUP Options

Option	Function
/W	Suppresses various informational messages BUP displays as backup operations are performed; for example, /W suppresses the *files processed* message (the listing of the files processed). The default operation is to display the messages. You might use this option with /Y if you are using BUP from a KMON command and IND control file to write a single disk or magtape output volume.
/X	Restores information that has been backed up using BUP.
/Y	Inhibits prompting for various responses otherwise required from the terminal. Allows using BUP from KMON command and IND control files to write a single disk or magtape output volume (mount prompts require terminal response). The default is to require terminal responses.
/Z	Initializes a volume for use as an output volume in a backup operation. You must explicitly initialize a magtape when using it as a backup volume. BUP automatically includes the initialization procedure for logical disks. Once you have initialized a backup volume, you can use it for any backup operation.

The following sections describe the preceding BUP options in detail. These sections are organized functionally rather than alphabetically.

Summary of BUP Operations

BUP can perform the following types of operations:

- Initialize a backup volume
- Make a backup saveset (container file) or subset (logical disk)
- Verify backed-up data
- Obtain directories of savesets and subsets (logical disks)
- Restore backed-up data

All of these operations can be done separately, and some can be combined with others. Table 3–2 lists the valid BUP option (CSI and DCL) combinations for these operations.

Note that a subset is an RT–11 logical-disk file, while a saveset is a container file for backed-up data.

Table 3–2: Valid BUP Option Combinations

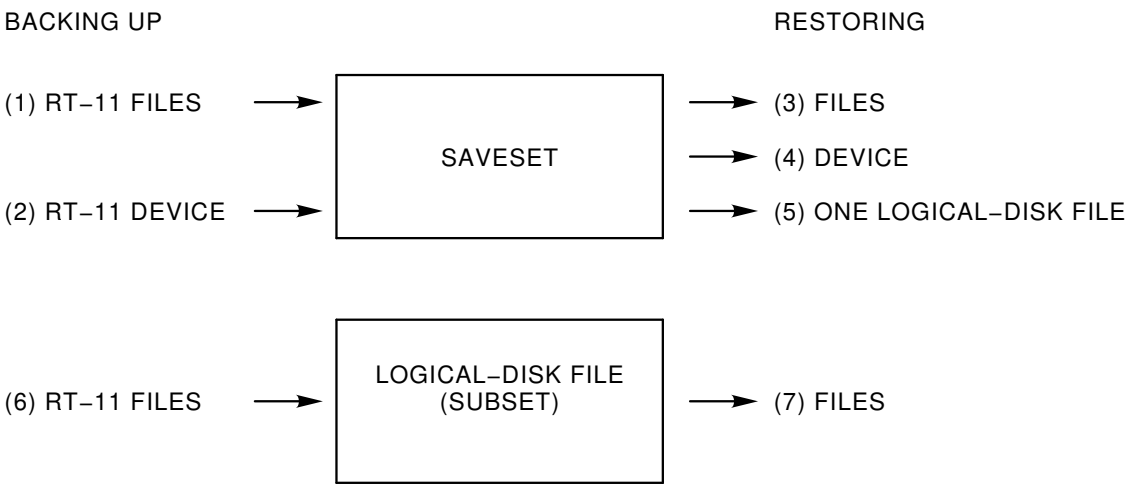
Operation	CSI Options	DCL Options
INITIALIZE	/Z	/INITIALIZE
	/Y	/NOQUERY
BACKUP to a saveset	/G	/NOSCAN
	/I	/DEVICE
	/M	/NOREWIND
	/V	/VERIFY
	/W	/NOLOG
	/Y	/NOQUERY
	/Z	/INITIALIZE
BACKUP to a subset	/R	/SUBSET
	/V	/VERIFY
VERIFY (only)	/X/V:ONL	/RESTORE/VERIFY:ONLY
DIRECTORY	/L	/DIRECTORY
		/DIRECTORY/OUTPUT[:filespec]
		/DIRECTORY/PRINTER
	/R	/SUBSET
	/S	/SAVESET

Table 3–2 (Cont.): Valid BUP Option Combinations

Operation	CSI Options	DCL Options
RESTORE from a saveset	/X	/RESTORE
	/E	/SYSTEM
	/F	/FILE
	/I	/DEVICE
	/M	/NOREWIND
	/S	/SAVESET
	/V	/VERIFY
	/W	/NOLOG
RESTORE from a subset	/R	/SUBSET
	/X	/RESTORE
	/E	/SYSTEM
	/V	/VERIFY
	/W	/NOLOG

Figure 3–1 summarizes BUP backup and restore operations. The command syntax following the figure lists the BUP operations indicated by the figure. The syntax is given first in CSI and then in DCL format.

Figure 3–1: Summary of the Backup Utility's Backup and Restore Operations



(1) out-dev:saveset-name=in-dev:file1[,file2,...]
BACKUP in-dev:file1[,file2...] out-dev:saveset-name

(2) out-dev:[saveset-name]=in-dev:/I
BACKUP/dev in-dev: out-dev:[saveset-name]

(3) out-dev:=in-dev:[saveset-name/S,]file1[,file2,...]
BACKUP/RESTORE in-dev:[saveset-name/SAVESET,]file1[,file2,...] out-dev:

(4) out-dev:=in-dev:[saveset-name]/I/X
BACKUP/RESTORE/DEVICE in-dev:saveset-name out-dev:

(4) out-dev:[file]=in-dev:saveset-name/F/X
BACKUP/RESTORE/FILE in-dev:saveset-name out-dev:[file]

(6) out-dev:[out-ldname]/R=in-dev:file1[,file2,...]
BACKUP in-dev:file1[,file2,...] out-dev:[out-ldname]/SUBSET

(7) out-dev:=in-dev:ldname.dsk/R,file1[,file2,...]/X
BACKUP/RESTORE in-dev:ldname.dsk/SUBSET,file1[file2,...] out-dev:

Backing Up Data

The next six sections describe the following aspects of backing up data:

- Steps of the backup operation
- Initializing backup volumes
- Verifying a data transfer
- Backing up files, volumes, and logical disks
- Backing up to logical disks (creating subsets rather than savesets)
- Backing up to magtapes and transporting them to the VMS operating system

Steps of the Backup Operation

The steps of the backup operation are the same for disks, diskettes, and magtapes:

1. BUP first asks you to mount the output volume in the device you have specified:

   ```
   Mount output volume in <device>; Continue?
   ```

   Type Y RETURN to continue or type N RETURN to abort the operation.

2. BUP then checks the type of output volume:

   - If the volume is already initialized by BUP and has space on it for new data, BUP displays the message:

     ```
     ?BUP-I-Appending to output volume 1
     ```

     In this case BUP does not initialize the volume. This allows you to have more than one backup saveset on a volume.

   - If the volume is not a valid RT–11 volume, BUP displays a message indicating that and allows you either to initialize it as a backup volume or to replace it.

   - If the volume is a random-access one that has not been initialized by BUP, BUP asks if you want to initialize the volume as a backup volume. If you respond NO, BUP prompts you to replace that volume with an already initialized backup volume, or to abort the operation.

     See the Initializing Backup Volumes (/Z) section for a description of the initialization procedure.

3. BUP copies the input file(s) or volume to the output volume. BUP lists the files as it copies them.

   When the output volume is full, BUP verifies the output data against the input volume (if you have specified that option). Then BUP prompts you to remove the backup volume and insert another if there is need for another volume. BUP repeats this process until all input has been copied.

4. When the backup operation is complete, BUP displays a message indicating that it has finished.

The output saveset file's creation date recorded in the directory is the system date during the backup operation. If no system date was set, no creation date is entered in the directory.

Initializing Backup Volumes (/Z)

You can initialize a volume in a separate operation or as part of a backup.

- To initialize a backup volume in a separate operation, use the following command syntax:

  out-device/Z

  where:

  out-device specifies the device containing the volume you want to initialize

- To initialize a backup volume as a part of the backup command, type Y RETURN at the initialization prompt. As a part of the backup operation, BUP automatically prompts you to initialize uninitialized backup volumes.

Preinitialization Procedure

Initialization overwrites any entries in a volume's directory. Therefore, BUP examines that volume and prompts you for confirmation before initializing it:

- If the backup volume has a standard RT–11 format (not formatted for BUP backup savesets), the volume may contain files that you wish to keep, and BUP prompts you with the message:

  ```
  ?BUP-W-Not a BACKUP volume <device>:
  <device>:/BUP Initialize; Are you sure?
  ```

  If you type Y RETURN, BUP proceeds to initialize the volume indicated by <device>. If you type N RETURN, BUP prompts you:

  ```
  Mount output volume in <device>; Continue?
  ```

  If you mount a new output volume and type Y RETURN, BUP continues with the initialization procedure. If you type N RETURN, BUP returns you to the asterisk prompt.

- If the volume already has backup saveset(s) and has space for more data, BUP does not initialize the volume. Rather, BUP displays the following message and backs up your data:

  ```
  ?BUP-I-Appending to output volume <number>
  ```

  where:

  <number> is the number BUP assigned to the backup volume when it placed the first backup data on it.

  If there is not enough space on the backup volume for all your data, BUP prompts you to mount another volume.

You can use the /Y option with /Z to suppress the confirmation messages.

Initialization Procedure

- For random-access volumes:
  - — /Z clears the directory of the volume and writes information into the home block (block 0) so BUP can recognize the volume as a backup volume.
  - — /Z scans the volume for bad blocks, since backup volumes must not contain bad blocks. If BUP finds a bad block on an output volume, BUP issues a fatal error message and stops the backup operation:

    ```
    ?BUP-I-Bad blocks detected; use another volume
    ```

    In this case, you must mount and initialize another volume.

    If you are sure all the backup media contain no bad blocks, you can include the /G option in your backup command to prevent a bad-block scan of each backup media.
- For magtapes:

  The /Z option rewinds the magtape and writes a volume and header label at the beginning of the magtape.
- For all backup volumes:

  When you back up files or volumes to more than one backup volume, BUP automatically prompts you for initializing each subsequent backup volume in the series.

The following command initializes a diskette as a backup volume:

```
*DU1:/Z [RET]
DU1:/BUP Initialize; Are you sure? Y [RET]
?BUP-I-Bad block scan started
?BUP-I-No bad blocks detected
```

Once you have initialized a backup volume, you should not reinitialize it to do further backup operations on that volume unless you want to delete backup information already on the volume.

To return a BUP-initialized volume to an RT–11 structure volume for use other than with BUP, initialize the volume using DUP. See Chapter 7 for a description of initializing volumes with DUP.

Verifying a Data Transfer (/V[:ONL])

The /V option verifies that output data matches input data in a backup/restore operation.

Three Verification Procedures

Depending on how you specify it, the /V option does one of three different verification procedures:

- **/V** verifies a data transfer as you are backing up the data.

  For each volume that is backed up, /V verifies that volume in one separate pass right after the data is backed up to that volume. The /V option compares the original to the backed-up data.

- **/X/V:ONL** verifies a data transfer after you back up the data but before you change or delete the original data.

  The /X/V:ONL option compares the backed-up data to the original. That is, /X /V:ONL first reads the data from the backed-up volume (as in a restore operation), then reads the original data and compares the two. It follows the restore procedure except for the actual restoring of the data.

  The /V:ONL option is valid only when used with the /X option and when the original data is as it was before a backup.

- **/X/V** verifies a data transfer as you are restoring the data.

  For each volume that is restored, /X/V verifies that volume, record for record, as it is being restored. Because this verification procedure is slightly different from /V and /X/V:ONL, /X/V is less sensitive to position errors (for example a slight slip in a magtape) than the other two verification procedures. It simply verifies that each restored data block can be correctly read.

Each verification procedure is alike in that they all:

- Check for device (read) errors.
- Check for data integrity of the blocks read.

The following command line includes the /V option to specify data verification.

```
*DU1:=DU0:LGFIL.DAT/V RET
```

Verification Messages

Depending on the type of verification, when BUP starts the verification process, BUP displays one of the following messages on the terminal:

```
?BUP-I-Verify pass started
?BUP-I-Restore/Verify operation started
```

Again, depending on the type of verification, if the verification is successful, BUP displays one of the next messages:

```
?BUP-I-Backup/Verify operation is completed
?BUP-I-Restore/Verify operation is completed
```

If the output data and the input data differ, BUP displays the error message:

```
?BUP-F-Verification error <dev:file.type>
```

NOTE

If your backup or restore operation involves magtapes, keep in mind that verification slows down the operation considerably.

Backing Up File(s)

When you specify no options in the BUP command line, BUP assumes you want to back up a file(s). This procedure allows you to back up all files on a device without copying empty blocks.

BUP allows you to use wildcards when backing up files. You can use wildcards to back up all files of a particular name or type, or to back up all files (*.*). Use the following command syntax to back up files:

out-device:[ssname]=in-device:file1[,file2,...]

where:

out-device	specifies the device in which you will mount the output volume(s) for the backup operation.
ssname	specifies the name you assign the saveset.
in-device	specifies the device and unit number of the volume containing the files to be backed up.
file(s)	specifies the file or files (wildcards allowed).

The following command illustrates the use of wildcards when backing up files:

```
*DU1:WRK=DL0:R*.FOR,*.MAC,T*.SAV RET
```

Backing Up Volumes (/I)

To back up an entire volume in image mode, use the /I option. This operation backs up everything on a volume, including empty blocks and home blocks. You can back up volumes to disks, diskettes, or magtapes. Use the following syntax to back up an entire volume:

out-device:[ssname]=in-device/I

where:

out-device specifies the device in which you will mount the output volume(s) for the backup operation.

ssname specifies the name you assign the saveset. If you specify no output file name, BUP uses the 2-letter mnemonic of the input device (for example, DU for DU1). The default output file type is BUP.

in-device specifies the device and unit number in which you will mount the volume to be backed up.

BUP copies the input volume to one or more output volumes. If there is more data than will fit on the output volume, BUP verifies the output data it has already copied (if you have specified that option) and then prompts you to remove the backup volume and insert another. BUP repeats this process until the entire input volume has been copied.

The following command backs up a DU0 volume to several diskettes using DU1. The backup volumes will contain the saveset file DU0.BUP when the backup operation is complete.

```
*DU1:=DU0:/I RET
```

Backing Up Logical Disks

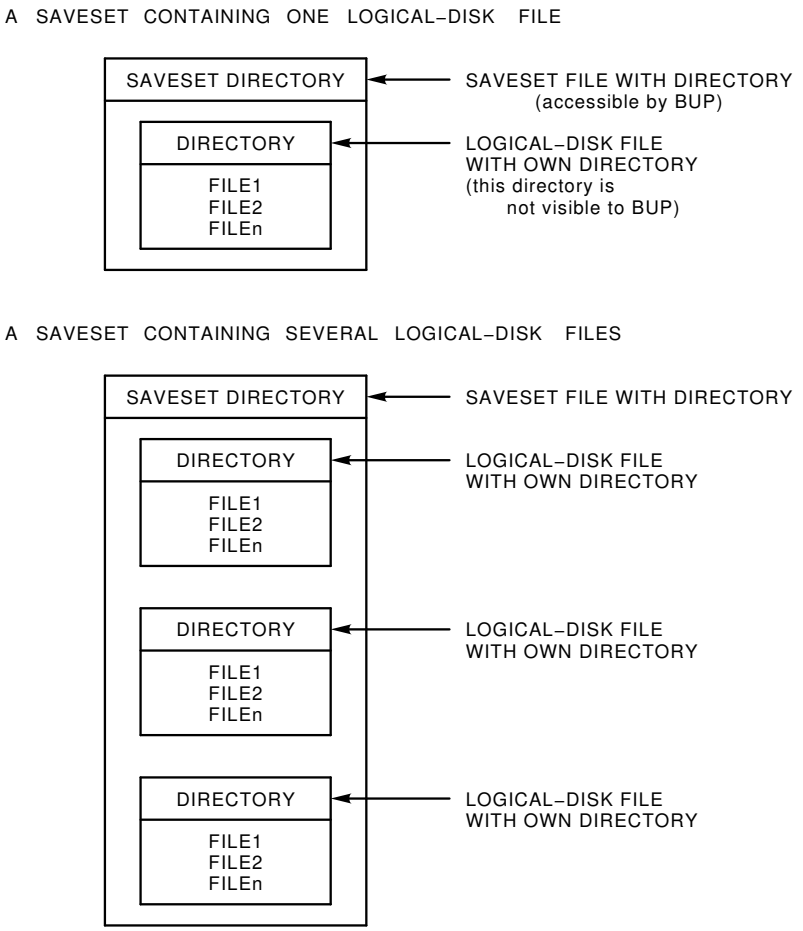
You can back up logical disks into saveset files or into subset (logical-disk) files:

- If you back up logical disks into saveset files, you have the following two choices:
  - — You can back them up as individual files, as described in the Backing Up Logical Disks as Files in Savesets section. In this case, the default name of the saveset is BACKUP.DSK.

    The advantage of doing this is that it can be convenient to save several logical disks in one saveset. The disadvantage is that you cannot access (get a directory listing or restore) individual files in each logical disk stored this way.

    Figure 3–2 illustrates backing up logical disks into saveset files.

Figure 3–2: Savesets Containing Logical Disks Backed Up as Files

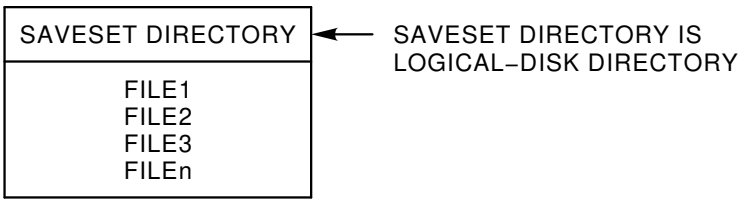


— You can back them up as device-image savesets, each saveset consisting of one logical disk. The Backing Up Logical Disks as Device Images in Savesets section describes how to do this. In this case, the default name of the saveset is the name of the logical-disk device from which you made the backup; for example, LD1.DSK.

The advantage of doing this is that you can access the files in the logical disk while it is in the saveset. That is, you can still get a directory listing of the files in the logical disk and restore them as individual files. The disadvantage is that you can store only one logical disk in a saveset to be able to access the files in the saveset.

Figure 3–3 shows a saveset with a logical disk backed up as a device image. Note that in this case, the logical-disk directory becomes the saveset directory; that is, they become one and the same directory.

Figure 3–3: Saveset Containing a Logical Disk Backed Up as a Device Image



- The Backing Up Files into Logical Disks (subsets) (/R) section describes how to back up files into logical disks. This is a feature of BUP when you use the /R option.

  The advantages of backing up information into logical disks (using the BUP /R option) are the following:

  — You perform no CREATE, INITIALIZE, or MOUNT operations; the /R option performs the equivalent of those operations for you.

  — The logical disk created by the /R option is identical to one you create manually, except no free blocks are allocated and the number of directory segments is only sufficient to contain the files being backed up. This makes for efficient storage of the information in logical disks and ease in accessing it, since it remains a logical disk.

  The disadvantages of backing up information into logical disks are the following:

  — The /R (subset) option is an alternative to the /S (saveset) option, and, unlike savesets, logical-disk images created by the /R option must reside on a single backup volume. Also, /R operations produce no bad-block scans (or mount prompts).

  — The /R option is appropriate only for file operations. You should not back up entire volumes (disk images) to logical disks with /R. You can use the DUP utility for that type of operation.

Backing Up Logical Disks as Files in Savesets

Backing up logical disks as files is the same as backing up any files with BUP as described in the Backing Up File(s) section.

It may be convenient to back up and restore logical disks as files. You can make periodic backups of multiple logical disk files, using wildcards, to a single saveset. For example, the following command syntax backs up all logical-disk files on volume *in-device* to saveset *ssname* on volume *out-device*:

out-device:ssname=in-device:*.DSK

Here and with any BUP wildcard operation, the files affected by the backup operation are listed on the terminal.

Backing Up Logical Disks as Device Images in Savesets

Backing up logical disks as device images is the same as backing up any volume with BUP as described in the Backing Up Volumes section.

Backing up logical disks as devices lets you obtain directories of backed-up logical disks, restore a logical disk as a device, or restore individual files from a backed-up logical disk.

To back up a logical-disk file as a device:

1. Associate the logical-disk file with a logical-disk unit. For example:

   ```
   .MOUNT LD0: DU1:MYWORK.DSK RET
   ```

2. Then back up the logical disk as associated with its logical-disk unit. The following command syntax backs up logical disk *LDn* to volume *out-device* as saveset *ssname*.

   ```
   *out-device:ssname=LDn:/I
   ```

Backing Up Files into Logical Disks (subsets) (/R)

The /R (subset) option creates logical-disk images of the files you want to back up. The option does this only on standard RT–11 disks, and not on BUP-formatted disks or any magtape volume. The standard RT–11 disk format is not changed, and the disk can continue to be used normally.

Use the following general command syntax to back up files to a logical-disk file:

out-device:[out-ldname]/R=in-device:file1[,file2,...]

where:

out-device	specifies the device on which the new logical disk file will appear. DK is the default output device.
out-ldname	specifies the file name you assign the new logical disk. The default output name is BACKUP and the default output file type is DSK.
in-device	specifies the device (with unit number) containing the file(s) you want in the logical disk. DK is the default input device.
file(s)	specifies the file or files (wildcards allowed) you want to copy.

For example, the following command backs up all files on DU0 of type OBJ to a logical disk, OBJ.DSK on device DU1. The success of the operation is verified by including the /V option in the command:

```
*DU1:OBJ/R=DU0:*.OBJ/V RET
```

Notice that file type .DSK is not included in the output specification (DU1:OBJ); DSK is the default file type. The command displays all files backed up to DU1. If DU1 does not contain sufficient free blocks for all the OBJ files, BUP returns an error message indicating insufficient space, and no files are backed up.

NOTE

The /R option, together with other BUP options, lets you obtain a directory of a logical disk and restore from a logical disk any files you specify. You can manually mount a logical disk created by the /R option and perform standard logical disk operations, such as copying, printing, and deleting files.

Backing Up to Magtapes

This section covers the following aspects of backing up to magtapes:

- Initializing magtapes
- Assigning unique names to savesets stored on magtapes
- Reading BUP magtapes on the VMS operating system

Initializing Magtapes

You must explicitly initialize any magtape volume before or during the first backup operation to that magtape (see the Initializing Backup Volumes section). When you back up files or volumes to a series of magtapes as part of a single backup operation, BUP implicitly initializes all subsequent magtapes in the series.

Assigning Unique Saveset Names

BUP normally rewinds magtapes before each backup operation. But if you intend to create a number of savesets on a magtape, you can prevent the magtape rewinding by including the /M (norewind) option in the backup command line. However, BUP cannot check that the saveset name you use is unique, unless the tape rewinds before each backup operation. So, Digital recommends that you explicitly assign unique saveset names, especially when you use the /M (norewind) option.

Reading BUP Magtapes on the VMS Operating System

You can transport BUP-written magtapes to a VMS system and extract files from those magtapes. Use the following procedure:

1. Use BUP to back up either files or a device image to a magtape. For example, the following RT–11 command backs up the device image LD3 to magtape Mdn, assigns the saveset name MYDISK to that backup image, and verifies the operation:

   ```
   .R BUP  RET
   *Mdn:MYDISK=LD3:/I/V  RET
   ```

2. Mount that backup magtape on a VMS system. The following VMS command mounts RTBUP (the backup magtage) as TAPE (a logical name) on device mddn. Note that all magtapes created by BUP have the volume label, RTBUP:

   ```
   $ MOUNT mddn: RTBUP TAPE  RET
   ```

3. Copy the BUP backup saveset to a VMS disk file. The following VMS command copies the file MYDISK from device mddn to a virtual-disk image file. MYDISK is the saveset name for the RT–11 device or files you backed up:

   ```
   $ COPY mddn:MYDISK.BUP *  RET
   ```

4. Use the VMS EXCHANGE utility to manipulate files on that virtual-disk file. See the VMS EXCHANGE utility documentation for information on using EXCHANGE. For example, the EXCHANGE commands do two things. The first

EXCHANGE command mounts the virtual-disk image file MYDISK.BUP on the virtual disk vdn. The second command displays a directory listing of device vdn:

```
$ EXCHANGE RET
EXCHANGE>MOUNT/VIRTUAL vdn: MYDISK.BUP RET
EXCHANGE>DIR vdn: RET
```

Listing Directories of Backed-Up Data

When you want to restore a saveset or a file located in a saveset, a directory of the savesets on your backup volumes can point you to the correct volume to mount. And a directory of the files in a saveset can point you to the saveset containing the files you want to restore.

BUP performs the following three types of directory operations:

- Lists savesets on a series of backup volumes
- Lists files in a saveset
- Lists files within a logical-disk file

The following sections describe these three operations.

Listing a Directory of Savesets on a Volume (/L)

You can list a directory of savesets on a backup volume by using the /L option or simply by the way you format the listing command (see the Command-Line Syntax section). The examples in the following sections illustrate both ways of listing directories.

Listing Volume Directories on Magtapes

You use the same command syntax to get volume directories on magtapes as on disks or diskettes. However, for certain magtape devices, this process can take some time; BUP must read to the logical end of magtape volumes before completing a directory listing.

Displaying Volume Directories

To display a backup volume directory, first mount the backup volume that contains either the entire saveset or the first section of a multivolume saveset. Then use either of the command formats shown in the following examples. Both commands display the directory of the backup volume in device DU1:

```
*DU1:/L RET
```

or

```
*,,TT:=DU1: RET
```

Printing Volume Directories

The following two commands send backup volume directories to a printer. You can use either command syntax to print the directories.

```
*,,LP:=DU1:/L RET
```

or

```
*,,LP:=DU1: RET
```

Storing Volume Directories in a File

The following two commands illustrate how to store backup volume directories in a file. You can use either of the two command formats. Both commands store a listing of all the savesets on the volume in DU1 and place that listing in a file on DU0:

```
*,,DU0:MYFILE.BUP=DU1:/L RET
```

or

```
*,,DU0:MYFILE=DU1: RET
```

Format of a Saveset Directory (a Listing of Savesets)

The directory structure of savesets on backup disks, diskettes, and magtapes is different from the standard RT–11 directory structure. In addition, a saveset directory on a magtape is different from one on a disk or diskette.

BUP disk, diskette, and magtape directories have the following information:

- **Saveset Name(s)**

  The saveset file name identifies a saveset section. If more than one saveset is on a volume, each one on that volume is listed.

- **Section Number(s)**

  A section of a saveset is the amount of the saveset that fits on one backup volume. So, the number of sections in a saveset is the same as the number of volumes used to back up a disk. For example, if a saveset is spread across more than one volume, the volume containing the first section of the saveset is identified as section 1, the second section as section 2, and so on.

  In summary, the section number in a saveset directory indicates which section of a saveset is on that volume.

  Disk and diskette directories have a second number following the saveset number and separated from it by a slash. The second number indicates how many sections a saveset file is divided into. For example, a 1/1 for saveset number information means the saveset is undivided and the entire saveset file is on that volume. However, a 1/2 means the saveset file is divided into 2 sections (since it did not fit on the volume), and the first section is contained on that backup volume.

  The second number is not on magtape saveset directories.

- **Blocks**

  The size of that saveset section in blocks on the backup volume followed by the total size of the saveset. If the two numbers are the same, the entire saveset is on that volume.

- **Date**

  The date on which that saveset was backed up to the backup volume.

Example Saveset Directory on a Diskette

The following example is a directory of the first volume in a series of five diskette backups that contains three savesets:

```
 RT-11 BACKUP
 05-May-91 09:28
 Volume 1

 Saveset      Section     Blocks       Date

OBJ   .BUP    1/1         474/474      05-May-91
TEMP  .BUP    1/1         14/14        05-May-91
RUNOFF.BUP    1/5         304/3114     05-May-91

 3 Saveset sections, 792 Blocks
 0 Free blocks
```

Example Saveset Directory on a Magtape

The following example is a directory of a BUP magtape. The magtape backup volume contains the second section (2348 blocks) of a 5400-block saveset named BIGDSK.BUP, the complete savesets FIRST.TXT and SECOND.BUP, and the first (408-block) section of a 988-block saveset THIRD.BUP.

```
 RT-11 BACKUP
 05-May-91 16:40

 Saveset      Section     Blocks       Date

BIGDSK.BUP    2           2348/5400    20-Mar-91
FIRST .TXT    1           800/800      20-Mar-91
SECOND.BUP    1           5400/5400    21-Mar-91
THIRD .BUP    1           409/988      26-Mar-91

 4 Saveset sections, 8957 Blocks
```

You would need to mount a previous magtape in this series to restore the saveset BIGDSK.BUP, because the first section of that saveset is not located on this volume. A printed directory of savesets for each magtape backup volume would direct you to the correct volume to mount. You can restore the savesets FIRST.TXT and SECOND.BUP from this volume. Proceed to the next magtape volume of this series to restore the second section of THIRD.BUP.

Listing a Directory of Files in a Saveset (/S/L)

To get a directory of a saveset, you must specify the /S option either with the /L option or with the command syntax described in the Command-Line Syntax section. The general syntax of the command with the /L option is:

in-device:[ssname]/S/L

where:

in-device specifies the device (with unit number) containing either the entire saveset or the first section of a multivolume saveset.

ssname specifies the saveset for which you want a file directory. If you do not specify a saveset name, you have two possibilities: on a random-access device, BUP looks for the saveset BACKUP.BUP; on a tape, BUP displays a directory of the first saveset on the tape.

For example, the following command displays the files backed up in saveset TEMP.BUP, residing on DU1:

```
*DU1:TEMP.BUP/S/L RET
```

You can also print the directory of files in a saveset or store the directory in a file. To do so, use the the /S option with the command syntax described in the Listing a Directory of Savesets on a Volume (/L) section.

Format of a File Directory (Listing of Files) in a Saveset

The following example shows the format of a file directory in a saveset:

```
 RT-11 BACKUP
 10-May-91 10:54
 Saveset: DU1:TEMP.BUP
 Created: Thursday   09-May-91 09:23

 File         Blocks  Volume          Date

TEMP  .TMP    2       1        Thursday   09-May-91
CACHE .TMP    3       1        Thursday   09-May-91

 2 Files, 5 Blocks
```

Listing the Files in a Logical Disk (/R/L)

Logical disks backed up with the /R option have the same format as regular logical disks. This means you can use BUP or DIR to display logical-disk file directories, whether or not BUP created the disks.

BUP enables you to list logical-disk directories without having to mount them separately. The following sections illustrate this.

Displaying a Logical-Disk Directory

Both of the following commands display a directory of the logical disk MYBACK.DSK on DU1:

```
*DU1:MYBACK.DSK/R/L RET
```

or

```
*,,=DU1:MYBACK.DSK/R RET
```

Printing a Logical-Disk Directory

Both of the following commands print a directory of the logical disk MYBACK.DSK on DU1:

```
*,,LP:=DU1:MYBACK.DSK/R/L RET
```

or

```
*,,LP:=DU1:MYBACK.DSK/R RET
```

Storing a Logical-Disk Directory in a File

Both of the following commands store a directory of the logical disk DU1:MYBACK.DSK in the file DU0:MYBACK.DIR:

```
*,,DU0:MYBACK.DIR=DU1:MYBACK.DSK/R/L RET
```

or

```
*,,DU0:MYBACK.DIR=DU1:MYBACK.DSK/R RET
```

Format of a Logical-Disk Directory Created by BUP

The directories of backed-up logical disks are similar to standard logical-disk directories. The following is an example directory listing of a logical-disk file as displayed by BUP:

```
 RT-11 BACKUP
 04-Jan-91 10:50
 Subset:  DU0:BACKUP.DSK

 File          Blocks            Date

PROG1 .OBJ     15          Friday     14-Dec-90
PROG2 .OBJ     234         Friday     14-Dec-90
PROG3 .OBJ     49          Monday     17-Dec-90
MEMO1 .TXT     10          Friday     04-Jan-91

 4 Files, 308 Blocks
```

Restoring Backed-Up Data (/X)

Use the /X option to restore backed-up data to a standard RT–11 formatted disk. You can restore:

- Complete savesets
- Selected files from a saveset
- Complete subsets (logical disks)
- Selected files from a subset

The next four sections describe the preceding four ways of restoring backed-up data.

You can use wildcards to restore files of only a particular name or type. If you use wildcards, or do not specifically name the files in a restore operation:

- BUP displays a list of the files it restores, as it restores them.
- You must use the /E option with the /X option to restore SYS (system) files.

Restoring Complete Savesets

You can restore complete savesets in the following three ways:

- By restoring all the files from a saveset
- By restoring a complete device image from a saveset
- By restoring a saveset as a logical-disk file

Restoring All the Files from a Saveset (ssname/S/X)

You can restore all the files in a saveset in one step by combining the /S option with the /X option to specify the name of the saveset you want to restore. Use the following general command syntax:

out-device=in-device:[ssname/S]/X

where:

ssname specifies the saveset you want to restore. If you do not specify a saveset name:

- On a random-access device, BUP looks for saveset BACKUP.BUP. If BUP does not find such a name, BUP returns an error message.
- On a magtape, BUP restores the files from the first saveset encountered on the magtape.

For example, the following command restores the saveset 28MAY.BUP from device DL0 to DL1, and verifies the restored data. As the files are restored from the saveset, they are listed on the terminal:

```
*DL1:=DL0:28MAY.BUP/S/X/V RET
```

See the Restoring Individual Files from Savesets section for more information on restoring files from savesets.

Restoring a Complete Device Image from a Saveset (/I/X)

Restoring a device image from a saveset means restoring the entire image of a volume with home blocks, directory, and any empty blocks to a standard RT–11 formatted disk. This means when you restore a device image to a disk, the disk is initialized as part of the operation.

On completion of this operation, BUP adjusts the output volume's RT–11 directory to correctly reflect the number of free blocks if the saveset was of a different size.

To restore a device-image saveset from a volume or series of volumes use the following general command syntax:

out-device=in-device:[ssname]/I/X

where:

ssname specifies the saveset you want to restore. If you do not specify the saveset name in the command line, BUP does one of the following:

- On a magtape backup volume, BUP restores the first saveset on the magtape.
- On a random-access device, BUP looks for a saveset matching the output device name. If BUP does not find such a name, BUP returns an error message. For example, the following command causes BUP to look for saveset DL1.BUP on device DL0:

```
*DL1:=DL0:/I/X/V RET
```

In the following command, MYDISK is the named saveset contained on magtape MS0. The command restores MYDISK to disk device DL1 and verifies the restoration.

```
*DL1:=MS0:MYDISK/X/I/V RET
```

If, in the preceding example, MYDISK is a saveset of an RX50 diskette, then, when the saveset is restored to the RL01/02 (DL) disk, the free blocks' value in the directory is adjusted to reflect the larger volume.

Restoring a Saveset as One Logical-Disk File (/F/X)

Restoring a saveset to a file means BUP copies the saveset from the backup volume as one file. This is helpful if your saveset file is a logical disk and you want to restore it as a file.

To restore a saveset as one file, use the /F option with the /X option in the following command syntax:

out-device:[filnam.ext]=in-device:ssname/F/X

where:

out-device specifies the volume to which you want the saveset restored.

filnam.ext specifies the name of the saveset when it is restored. If you specify no name and file type, BUP gives it the ssname you specified with the file type BUP.

in-device specifies the backup volume containing the saveset you want to restore.

ssname specifies the saveset.

Restoring Individual Files from Savesets

Use the following general command syntax to restore files from savesets:

out-device=in-device:[ssname/S,]file1[,file2,...]/X

where:

ssname specifies the saveset containing your file. The saveset can contain an entire device image or only individual files.

Depending on your input device, one of two things can happen if you do not specify the saveset name (with /S) in the command line:

- If the input device is a magtape, BUP attempts to restore the specified file or files from the first saveset encountered on the magtape.
- If the input device is a random-access one, BUP looks for a saveset named BACKUP.BUP. If BUP does not find BACKUP.BUP, it returns an error message. For example, the following command causes BUP to look for saveset BACKUP.BUP on device DL0:

```
*DL1:=DL0:/X/V RET
```

file(s) specifies the file(s) you want to restore. Use commas to separate files, when you specify more than one. You can also use wildcards to restore files of a particular name or type or to restore all files (*.*).

Examples

1. Assuming the saveset 28MAY.BUP contains the file FOO.OBJ, you can restore that file to device DL1 and verify the restoration by using the following command:

   ```
   *DL1:=DL0:28MAY.BUP/S,FOO.OBJ/X/V RET
   ```

2. You could also restore and verify all files of type OBJ from saveset 28MAY.BUP, using the following command:

   ```
   *DL1:=DL0:28MAY.BUP/S,*.OBJ/X/V RET
   ```

Restoring Logical Disks

The operations you choose to restore data depend on which way the data was backed up.

You can back up logical disks mainly in two ways: as savesets or as subsets; and if you back them up as savesets, you can store them as files or as device images. So, the phrase *restoring a logical disk* can apply to any one of the following BUP operations:

- Restoring a logical-disk saveset as a device image
- Restoring a logical-disk saveset as a file
- Restoring a logical-disk file from a saveset containing several logical-disk files
- Restoring files from a device-image saveset of a logical disk
- Extracting one or more files from a logical disk (a subset)

The following subsections briefly describe the preceding logical-disk saveset operations, while the Extracting File(s) from a Logical Disk (Subset) /R/X section describes the subset operation.

Restoring a Logical-Disk Saveset as a Device Image

The commands for restoring a logical-disk saveset as a device image are the same as for any device-image restoration (see the Backing Up Volumes (/I) section).

NOTE

Although this case restores a a logical disk, the /I in the command initializes the output device. That is, the logical-disk image is written to the output volume on a block-for-block basis, starting at block 0.

The next command syntax restores a logical-disk *LDn*, residing on backup volume *in-device*, to volume *out-device*.

out-device:=in-device:LDn/X/I

In the following command, OLD.DSK is the named logical-disk saveset contained on device DL1. The command restores OLD.DSK to disk device DL0 and verifies the restoration:

```
*DL0:=DL1:OLD.DSK/X/I/V RET
```

Restoring a Logical-Disk Saveset as One File

The commands for restoring a logical-disk saveset as one file are the same as for restoring a saveset as a whole to a file image (see the Restoring a Saveset as One Logical-Disk File (/F/X) section).

Restoring a logical disk as a single file image writes the entire logical disk as a file to the output volume. For example, in the following command syntax, *in-device* is

the backup volume containing the logical-disk file and *out-device* is the volume to which the logical-disk file is restored:

out-device:[filnam.DSK]=in-device:filnam.DSK/X/F

Restoring a Logical-Disk File from a Multiple-File Saveset

The commands for restoring a logical-disk file from a multiple-file saveset are the same as those for restoring individual files from savesets (see the Restoring Individual Files from Savesets section).

For example, in the following command, FILES.DSK is the saveset on device DL1 that contains several logical disks. The command restores the logical disk MONDAY.DSK from the saveset FILES.DSK to disk device DL0 and verifies the restoration:

```
*DL0:=DL1:FILES.DSK/S,MONDAY.DSK/X/V RET
```

Restoring Individual Files from a Device-Image Saveset of a Logical Disk

Restoring individual files from a device-image saveset of a logical disk is the same as restoring individual files from savesets (see the Restoring Individual Files from Savesets section).

The following command syntax restores files **.ext* from saveset *LDn*, residing on backup volume *in-device*, to volume *out-device*:

out-device:=in-device:LDn/S,*.ext/X

Extracting File(s) from a Logical Disk (Subset) /R/X

Extracting a file from a logical-disk subset means getting a copy of the file from the logical disk. You can extract (or restore) one or more selected files from mounted or unmounted logical disks by using the following general command syntax:

out-device:=in-device:ldname.dsk/R,filename1[,filename2,...]/X

where:

out-device	specifies the volume to which you want the file(s) restored.
in-device	specifies the volume containing the logical disk.
ldname.dsk	specifies the name of the logical disk.
filename	specifies the file or files you want to restore from the logical disk. You can use wildcards.

Examples

1. The following command extracts and verifies the file FOO.OBJ in logical disk OBJ.DSK on DL1, to device DL0:

   ```
   *DL0:=DL1:OBJ.DSK/R,FOO.OBJ/X/V RET
   ```

2. You can also extract multiple selected files from a logical disk, using wildcards. For example, the following command extracts all files of the name WOOGA and verifies the operation:

   ```
   *DL0:=DL1:OBJ.DSK/R,WOOGA.*/X/V RET
   ```

3. This example extracts a copy of all files:

   ```
   *DL0:=DL1:OBJ.DSK/R,*.*/X/V RET
   ```

Restoring Files Backed Up Prior to RT–11 Version 5.5

Versions of RT–11 prior to Version 5.5 let you create a file-image backup that was not contained within a saveset. Such a file image has a format different from that of a saveset. You restore such a file image from a backup volume or series of backup volumes by including the /F (FILE) option. Because you are performing a file restoration to a disk, BUP does not initialize that disk as part of the operation.

Use the following general command syntax to restore this type of file:

out-device:[filnam.ext]=in-device:filnam.ext/X/F

where:

out-device	specifies the volume to which you want the file restored.
filnam.ext	specifies the file you want restored. If you do not specify it with the output device, BUP gives it the name it has on the input device.
in-device	specifies the backup volume containing the file you want to restore.

For example, the following command restores, with verification, the file image FIRST.TXT from magtape MS0 to device DL1:

```
*DL1:=MS0:FIRST.TXT/X/F/V RET
```

DCL Equivalents of BUP Utility Operations

Table 3–3 lists the DCL BACKUP command options that are equivalent to BUP utility operations.

The first part of the table lists that part of the CSI BUP command syntax that is equivalent to three different DCL BACKUP/DIRECTORY options. The rest of the table alphabetically lists all the BUP options having DCL equivalents.

Table 3–3: DCL Equivalents of BUP Utility Operations

CSI Command/Option	DCL Option
,,TT:= (3rd output filespec)	/DIRECTORY
,,filespec=	/DIRECTORY/OUTPUT:filespec
,,LP:=	/DIRECTORY/PRINTER
/E	/SYSTEM
/F	/FILES
/G	/NOSCAN
/I	/DEVICE
/L*	/DIRECTORY
/M	/NOREWIND
/R	/SUBSET
/S	/SAVESET
/V[:ONL]	/VERIFY[:ONLY]
/W	/NOLOG
/X	/RESTORE
/Y	/NOQUERY
/Z	/INITIALIZE

*This option exists for compatibility with previous versions of BUP.

Chapter 4

CONFIG, CONSOL, and DATIME Utilities

This chapter describes the following three unsupported utilities:

- CONFIG
- CONSOL
- DATIME

Configuration Utility (CONFIG)

The Configuration Utility (CONFIG) is an unsupported utility that enables you to determine:

- Whether a specified handler is installed.
- Whether a specified memory location exists in a system.
- Whether the contents of a specified location match a specific value.
- Whether an MSCP device unit contains fixed or removable media and whether that media is available.

Do not confuse this utility with the configuration procedure contained within the IND control file CONFIG.COM. You run that control file with the command IND CONFIG, and its purpose is to delete unnecessary distributed files from your working system disk. See the *RT–11 Automatic Installation Guide* for a brief description of CONFIG.COM, the IND configuration procedure.

To use CONFIG, you must have the file CONFIG.SAV on your system device.

CONFIG Command-Line Syntax

The syntax for the CONFIG command line changes somewhat depending on what information is to be returned by the utility:

- To check on devices, use the syntax:

  CONFIG dev:[/option1/option2...]

- To check on memory locations, use the syntax:

  CONFIG /option1[/option2...]

CONFIG Options

Table 4–1 alphabetically lists the CONFIG options.

Table 4–1: CONFIG Options

Option	Type	Function
/A:addr	Memory-Location	Determines whether memory location addr exists. Useful for finding out how much memory a system includes. If a read of location addr succeeds, USERRB is set to 1. If a read causes a trap, USERRB is set to 10 (for fatal error).
/B	Memory-Location	Use with /A operations to perform a byte operation instead of a word operation.
/D[:yes][:no]	Memory-Location	The /D option with no optional parameter causes CONFIG to return an error message in addition to setting bits in USERRB. You can assume success if no error message is returned. Using /D:yes causes CONFIG to continue to return error messages without repeating the /D option. /D:no turns off /D:yes.
/F:addr	Memory-Location	Determines whether file location addr exists. If a read of location addr succeeds, USERRB is set to 1. If the location (addr) does not exist in the file, USERRB is set to 10 (for fatal error). The /F option lets you use CONFIG comparison options (/V, /M, and /B) with files. Also, using /F with a file locks the specified file to CONFIG, which lets you access the file using only the /F option (without specifying the file again), and increases the access speed.
/M:mask	Memory-Location	Use with /A and /V:contents to test bits within the memory location specified with /A. The variable, mask, specifies a bit mask that specifies which bits to test.
/P	Device	Checks physical-device names and ignores logical-device names.
/R:offset	Memory-Location	Use with /A to specify locations based on an offset—from the beginning of RMON (offset) rather than an actual memory address (addr).

Table 4–1 (Cont.): CONFIG Options

Option	Type	Function
/T	Device	Use the /T option to determine if an MSCP device unit contains fixed or removable media and whether that media is available. Use the following command syntax: **RUN CONFIG dev:/T:type** where: **dev** is the MSCP device unit. **type** is REM or FIX. Specify REM for type if you want CONFIG to logically assume the MSCP device media is removable. Specify FIX for type if you want CONFIG to assume the MSCP device media is fixed. The following example tests MSCP device DU3 and assumes the media in DU3 is removable: `.RUN CONFIG DU3:/T:REM` RET If the media in DU3 is removable, the user error byte (USERRB, byte 53 in the system communications area) is set to 1 for success. If the media is not removable, USERRB is set to 4 for error. If DU3 is not available, USERRB is set to 10 for fatal error.
/V:contents	Memory-Location	Use with /A to verify that the contents of the specified location equal the value contents. If the contents of the location match the value contents, USERRB is set to 1. If they do not match, USERRB is set to 4. If accessing the location causes a trap (the location does not exist), USERRB is set to 10.

CONFIG Examples

1. To determine whether a handler is installed, issue the following command:

   ```
   CONFIG dev:
   ```

   where *dev* is the handler's physical or logical device name; for example:

   ```
   .CONFIG LD: RET
   ```

   If the handler is installed, USERRB (memory location 53 in the system communication area) is set to 1 for success. If the handler is not installed, USERRB is set to 4 for error.

2. To check only physical device names (and ignore logical names), use the /P option:

   ```
   .CONFIG dev:/P RET
   ```

   For example, the following CONFIG command determines whether the LS handler is installed. Since the option /P is included in the command, CONFIG searches for only the physical device name LS and not for devices whose logical name is LS:

   ```
   .CONFIG LS:/P RET
   ```

3. To check information about memory locations, type the following command:

   ```
   .CONFIG /option[.../option] RET
   ```

   where *option* specifies one or more of the CONFIG memory-location options.

   The following command asks CONFIG to determine whether location 177776 exists, and tests whether the high 8 bits match the value 210:

   ```
   .RUN CONFIG /A:177776/V:104000/M:177400 RET
   ```

Console Utility (CONSOL)

The Console Utility (CONSOL) is an unsupported utility that changes the system console (having a local DL interface) on systems that do not include multiterminal support. CONSOL relocates the monitor console from one local DL line to another. CONSOL makes only in-memory changes so that the monitor reverts to the boot-time console at the next reboot.

To use the CONSOL utility, type:

```
.RUN CONSOL RET
```

CONSOL requires no further commands or interaction.

Depending on your hardware configuration, it may be necessary to edit CONSOL.MAC to reflect the correct CSR and vector of the new system console. In this case, you must also rebuild (reassemble and relink) CONSOL.SAV.

Datime Utility (DATIME)

The Datime Utility (DATIME) is an unsupported utility, usually used in STRxx.COM files, that forces entry of the current date and time.

There are two versions of this utility:

- DATIME.COM (an IND control-file procedure)
- DATIME.SAV (a runnable save image)

Both versions perform the same function.

You can modify DATIME.COM, but DATIME.COM requires that the file IND.SAV be on the system disk. Therefore, when running from small media, you may need to use DATIME.SAV.

To use DATIME, include one of the following commands in your STRxx.COM file:

```
IND DATIME
```

or

```
R DATIME
```

Chapter 5
Directory Utility (DIR)

The Directory Utility (DIR) lists a wide range of directory information. It can list directory information about a specific device, either in summarized form—where only the number of files stored per segment is given—or in more detailed form—where file names, file types, creation dates, and other file information is given. DIR can organize its listings in several ways, such as alphabetically or chronologically.

Calling and Terminating DIR

To call DIR from the system device, respond to the dot prompt (.) displayed by the keyboard monitor by typing:

```
.R DIR RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to enter a command string. If you press only RETURN in response to the asterisk, DIR displays its current version number.

You can type CTRL/C to terminate DIR and return control to the monitor when DIR is waiting for input from the console terminal. You must type CTRL/C twice to abort DIR at any other time. To restart DIR, type R DIR or REENTER in response to the monitor's prompt.

Reading Directory Listings

Directory listings normally display on the terminal in two columns. Read the entries across the columns, moving from left to right, one row at a time. Directory listings that are sorted, however, are an exception to this. (Sorted directories are produced by /A, /R, and /S options.) Read these listings by reading the left column from top to bottom, then reading the right column from top to bottom.

Command-Line Syntax

The Command-String Interpreter (CSI) Language section in Chapter 1 describes the general syntax of a command line that DIR accepts.

Specifying Parameters

You can specify only one input and one output device, but you can specify up to six file names on the input device. The default device for output is the terminal. The default file type for an output file is .DIR. The default device for input is DK. If you omit the input specification completely, DIR uses DK:*.*. If you do not supply an option, DIR performs the /L operation.

NOTES

Wildcards are valid with DIR for the input specification only.

Unless otherwise indicated, numeric arguments are interpreted as octal. Remember to put a decimal point after a decimal number to distinguish it from an octal number.

Specifying a DIR Option with a Date

Some of the DIR options accept a date as an argument in the command line. The syntax for specifying the date is:

dd:mmm:yy

where:

dd	the day (a decimal integer in the range 1–31)
mmm	the month (the first three characters of the name of the month)
yy	the year (a decimal integer in the range 73–99)

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, DIR displays -BAD- in the creation date column of each file created beyond the end-of-month.

NOTE

You can eliminate a -BAD- entry by using the RENAME /SETDATE command after you have set the date.

DIR Option Descriptions

Alphabetical Option (/A)

The /A option lists the directory of the device you specify in alphabetical order by file name and type. Note that /A sorts numbers after letters. It has the same effect as the /S:NAM option. The following example lists the directory of device DU0 in alphabetical order:

```
*DU0:/A RET
 14-Mar-91
BUILD .SAV   100  06-Sep-90      SWAP  .SYS   25  05-Dec-90
DATE  .TXT     3  06-Sep-90      SYSMAC.MAC   41  19-Nov-90
MYPROG.MAC    36P 12-Oct-90      TM    .MAC   25  27-Nov-90
RFUNCT.SYS     4  19-Nov-90      TT    .SYS    2  19-Nov-90
RT11SB.SYS    67  19-Nov-90      VTMAC .MAC    7  19-Nov-90
 10 Files, 306 Blocks
 180 Free Blocks
```

Block Number Option (/B)

The /B option includes the starting block number in decimal of all the files listed in a directory of the volume you specify. The following example lists the directory of device DU0, including the starting block numbers of files:

```
*DU0:/B RET
 14-Jan-91
FSM   .MAC    31P 19-Nov-90    2955      BATCH .MAC   102P 19-Nov-90    2986
ELCOPY.MAC     8P 19-Nov-90    3090      ELINIT.MAC    15P 19-Nov-90    3096
ELTASK.MAC    15P 19-Nov-90    3111      ERROUT.MAC    48P 19-Nov-90    3126
ERRTXT.MAC     9P 19-Nov-90    3174      SYCND .BL      3P 19-Nov-90    3183
SYSTBL.BL      4P 19-Nov-90    3186      SYCND .DIS     5P 19-Nov-90    3190
SYSTBL.DIS     4P 19-Nov-90    3195      SYCND .HD      5P 19-Nov-90    3199
ABSLOD.SAV    48  15-MAR-90    3204      CHESS .SAV    40  17-Aug-90    3252
PETAL .SAV    36  11-Sep-90    3292      LAMP  .SAV    29  16-Mar-90    3328
WUMPUS.SAV    30  16-Mar-90    3357
 17 Files, 348 Blocks
 138 Free blocks
```

Columns Option (/C[:value])

The /C[:value] option lists the directory in the number of columns you specify. The *value* argument represents an integer in the range 1–9. If you do not use the /C[:value] option, DIR lists the directory in two columns for normal listings and five columns for abbreviated listings. The following command, for example, lists the directory of device DU1 in one column:

```
*DU1:/C:1 RET
 4-Jan-91
SWAP  .SYS    25P 19-Nov-90
RT11SB.SYS    67P 19-Nov-90
RT11FB.SYS    80P 19-Nov-90
LETTER.TXT    64P 19-Nov-90
TT    .SYS     2P 19-Nov-90
MEMO2 .TXT     3P 19-Nov-90
MEMO1 .TXT     3P 19-Nov-90
 7 Files, 244 Blocks
 242 Free blocks
```

DIR Option Descriptions

Date Option (/D[:date])

The /D[:date] option includes in the directory listing only those files having the date you specify. The default date is the system's current date. For example, the following command lists all the files created on January 14, 1991:

```
*DU0:/D:14.:JAN:91. RET
 15-Jan-91
RT11SB.SYS    67P 14-Jan-91      RT11FB.SYS    80P 14-Jan-91
LETTER.TXT    63P 14-Jan-91      DX    .SYS     3P 14-Jan-91
SWAP  .SYS    25P 14-Jan-91      TT    .SYS     2P 14-Jan-91
MEMO1 .TXT     3P 14-Jan-91      DATE  .TXT     4P 14-Jan-91
LP    .SYS     2P 14-Jan-91      PIP   .SAV    16  14-Jan-91
DUP   .SAV    41  14-Jan-91      RESORC.SAV    15  14-Jan-91
DIR   .SAV    17  14-Jan-91      RK    .SYS     3  14-Jan-91
EDIT  .SAV    19  14-Jan-91      DD    .SYS     5  14-Jan-91
SRCCOM.SAV    13  14-Jan-91      BINCOM.SAV    11  14-Jan-91
SLP   .SAV     9  14-Jan-91      SIPP  .SAV    14  14-Jan-91
 20 Files, 412 Blocks
 73 Free blocks
```

Entire Option (/E)

The /E option lists the entire directory including the unused areas and their sizes in blocks (decimal). Use it to find free space before you extend a file (with the monitor CREATE command or DUP /C option). The following example lists the entire directory of device DU1, including unused areas:

```
*DU1:/E RET
 20-Mar-91
SWAP  .SYS    25P 23-Oct-90      RT11SB.SYS    67P 23-Oct-90
RT11FB.SYS    80P 19-Oct-90      LETTER.TXT    64P 19-Oct-90
TT    .SYS     2P 19-Oct-90      MEMO2 .TXT     3P 19-Oct-90
MEMO1 .TXT     3P 23-Oct-90      DX    .SYS     3P 19-Oct-90
DATE  .TXT     4P 19-Nov-90      RF    .SYS     3P 19-Nov-90
RK    .SYS     3P 19-Nov-90      DL    .SYS     4P 23-Oct-90
DM    .SYS     5P 23-Oct-90      DS    .SYS     3P 19-Nov-90
DD    .SYS     5P 23-Oct-90      LP    .SYS     2P 23-Oct-90
LS    .SYS     2P 19-Nov-90      CR    .SYS     3P 19-Nov-90
MS    .SYS     9P 27-Nov-90      MTHD  .SYS     3P 23-Oct-90
DISMT1.COM     9P 27-Nov-90      MMHD  .SYS     4P 19-Nov-90
NUMBER.PAS     1  11-Dec-90      TONY  .AGP    14  17-Aug-90
NUM3  .LST     1  13-Dec-90      < UNUSED >   565
 25 Files, 322 Blocks
 164 Free blocks
```

Fast Option (/F)

The /F option lists only file names and file types, omitting file lengths and associated dates. For example, the following command lists only file names and types from device DU0:

```
*DU0:/F RET
 16-Aug-90
DATE  .TXT      PIP  .SAV      DIR   .SAV      DUP   .SAV      SWAP  .SYS
RT11SB.SYS      RT11FB.SYS     LETTER.TXT      TT    .SYS      MEMO2 .TXT
 10 Files, 312 Blocks
 174 Free blocks
```

Begin Option (/G)

The /G option lists the directory of the volume you specify, beginning with the file you specify and including all the files that follow it in the directory.

Usually, the disk you are using as a system device contains a number of files the operating system needs. These files include .SYS monitor files, .SAV utility program files, and various .OBJ, .MAC, and .BAK files. They are generally grouped together and usually listed at the beginning of a normal volume directory. Files that you create and use, such as source files and text files, are also generally grouped together and follow the operating system files in the directory. If you specify the name of the last system file with the /G in the command line, DIR displays a directory of only those files that you created and stored on the volume.

The following command, for example, lists the last system file (CT.SYS) and all the user files that follow it:

```
*DU0:CT.SYS/G RET
 10-Jan-91
CT    .SYS     5  10-Aug-90      DIR   .SAV    17  03-Aug-90
RK    .SYS     3  13-Aug-90      EDIT  .SAV    19  03-Aug-90
STARTS.COM     1  27-Aug-90      DD    .SYS     5  19-Aug-90
SRCCOM.SAV    13  13-Aug-90      BINCOM.SAV    11  05-Oct-90
SLP   .SAV     9  13-Aug-90      SIPP  .SAV    14  05-Oct-90
 10 Files, 107 Blocks
 73 Free blocks
```

Since Option (/J[:date])

The /J[:date] option lists a directory of all files stored on the device you specify created on or after the date you supply. The default date is the system's current date. The following command lists all files on device DU0 created on or after January 20, 1991:

```
*DU0:/J:20.:JAN:91. RET
 20-Mar-91
RT11SB.SYS   67P 28-Jan-91      RT11FB.SYS    80P 02-Feb-91
LETTER.TXT   63P 19-Feb-91      DX    .SYS     3P 10-Mar-91
SWAP  .SYS   25P 02-Feb-91      TT    .SYS     2P 15-Mar-91
SIPP  .SAV   14  02-Feb-91
 7 Files, 154 Blocks
 332 Free blocks
```

Before Option (/K[:date])

The /K[:date] option displays a directory of files created before the date you specify. The default date is the system's current date. The following command lists all files stored on device DU1: created before March 15, 1991:

```
*DU1:/K:15.:MAR:91. RET
 20-Mar-91
FORTRA.SAV    191  14-Mar-91      BASIC .SAV    51  25-Feb-91
 2 Files, 242 Blocks
 38 Free blocks
```

DIR Option Descriptions

Listing Option (/L)

The /L option lists the directory of the volume you specify. The listing contains the current date, all files and their associated creation dates, the number of blocks used by each file, total free blocks on the device (if disk), the number of files listed, and the total number of blocks used by the files. File lengths, number of blocks, and number of files are indicated as decimal values. For example, the following command lists on the line printer the directory for device DU1:

```
*LP:=DU1:/L RET
```

The printer output looks like this:

```
 20-Nov-90
RT11SB.SYS    67P 03-Jul-90      RT11FB.SYS    80P 13-Aug-90
LETTER.TXT    63P 15-Mar-90      DX    .SYS     3P 13-Aug-90
SWAP  .SYS    25P 13-Aug-90      TT    .SYS     2P 13-Aug-90
MEMO1 .TXT     3P 13-Aug-90      DATE  .TXT     4P 13-Aug-90
LP    .SYS     2P 20-Nov-90      PIP   .SAV    16  25-Jul-90
DUP   .SAV    41  26-Mar-90      RESORC.SAV    15  13-Aug-90
EDIT  .SAV    19  13-Aug-90      STARTS.COM     1  27-Aug-90
SIPP  .SAV    14  13-Aug-90
 15 Files, 413 Blocks
 73 Free blocks
```

Note that if you specify no options in the command string, this is the default directory operation.

Unused Areas Option (/M)

The /M option lists only a directory of unused areas and their size on the volume you specify. For example, the following command lists all the unused areas on device DL0:

```
*DL0:/M RET
 14-Dec-90
< UNUSED >    11                 < UNUSED >     2
< UNUSED >    26                 < UNUSED >    32
< UNUSED >     1                 < UNUSED >   525
< UNUSED >     0                 < UNUSED >   565
 0 Files, 0 Blocks
 1162 Free blocks
```

Summary Option (/N)

The /N option lists a summary of the volume directory. The summary lists the number of files in each directory segment and the number of segments in use on the volume you specify. The segments are listed in the order in which they are linked on the volume.

The following command lists the summary of the directory for device DK:

```
*/N RET
 14-Jan-91

    44 Files in segment 1

    46 Files in segment 4

    37 Files in segment 2
```

```
    34 Files in segment 5

    38 Files in segment 3

    16 Available segments, 5 in use

 199 Files, 3647 Blocks
 1115 Free blocks
```

Octal Option (/O)

The /O option is similar to the /L option, but lists the sizes (and starting block numbers if you use /B) of the files in octal. If the device you specify is a magnetic tape, DIR displays the sequence number in octal. For example, the following command lists the directory of device DU0, with sizes in octal:

```
*DU0:/O RET
 14-Jan-91 Octal
MYPROG.MAC    44P 12-Nov-90      TM    .MAC     31  27-Nov-90
VTMAC .MAC     7  18-Oct-90      SYSMAC.MAC     51  19-Nov-90
SWAP  .SYS    31  05-Sep-90      ANTON .MAC      4  19-Nov-90
RT11SB.SYS   103  19-Nov-90      TT    .SYS      2  19-Nov-90
DX    .SYS     3  29-Aug-90      BUILD .MAC    144  19-Nov-90
 10 Files, 462 Blocks
 264 Free blocks
```

Exclude Option (/P)

The /P option lists a directory of all files on a volume, excluding those that you specify. You may specify up to six files:

```
*DU1:*.SAV/P RET
 29-Feb-91
RT11SB.MAC    67P 06-Jan-91      RT11FB.MAC     80P 06-Jan-91
RT11BL.MAC    63P 06-Jan-91      DU    .MAC      3P 06-Jan-91
SWAP  .MAC    25P 06-Jan-91      TT    .MAC      2P 06-Jan-91
DP    .MAC     3P 06-Jan-91      DU    .MAC      4P 06-Jan-91
LP    .MAC     2P 06-Jan-91      RK    .MAC      3  06-Jan-91
STARTS.COM     1  27-Jan-91      DD    .MAC      5  06-Jan-91
 12 Files, 258 Blocks
 73 Free blocks
```

This command lists all files on device DU1 except SAV files.

Deleted Option (/Q)

The /Q option lists a directory of the volume you specify, listing the file names, types, sizes, creation dates, and starting block numbers in decimal of files that have been deleted but whose file name information has not been destroyed. The file names that display represent either tentative files or files that have been deleted. This can be useful in recovering files that have been accidentally deleted. Once you identify the file name and location, you can use DUP to rename the area. See the description of the CREATE (T:value) option of the DUP utility for an explanation of how to do this:

```
*DISK.DIR=/Q RET
```

DIR Option Descriptions

This command creates a file called DISK.DIR on device DK that contains directory information about unused areas from device DK. Use the monitor TYPE command to read the file:

```
.TYPE DISK.DIR/LOG RET
 Files copied:
DK:DISK.DIR     to TT:
 12-Oct-90
EXAMPL.FOR    23  03-Sep-90   1403      MTHD  .SMP     5  09-Sep-90    2915
SCOPE .PIC     3  22-Sep-90   2926
 0 Files, 0 Blocks
 0 Free blocks
```

Reverse Option (/R)

The /R option lists a directory in the reverse order of the sort you specify with the /A or /S option. This command lists the directory of device DU0 in reverse file size order (from largest to smallest):

```
*DU0:/S:SIZ/R RET
 14-Jan-91
BUILD .MAC   100  06-Sep-90      TM    .MAC    25  27-Nov-90
RT11SB.SYS    67  19-Nov-90      VTMAC .MAC     7  19-Nov-90
SYSMAC.MAC    41  19-Nov-90      RFUNCT.SYS     4  19-Nov-90
MYPROG.MAC    36P 12-Oct-90      DX    .SYS     3  06-Sep-90
SWAP  .SYS    25  05-Dec-90      TT    .SYS     2  19-Nov-90
 10 Files, 306 Blocks
 180 Free blocks
```

Sort Option (/S[:category])

The /S[:category] option sorts the directory of the specified volume according to a three-character code you specify as :category. The following table summarizes the codes and their functions.

Sort Codes

Code	Function
DAT	Chronological by creation date. Files that have the same date are sorted alphabetically by file name and file type.
NAM	Alphabetical by file name. Files that have the same file name are sorted alphabetically by file type (this has the same effect as the /A option).
POS	According to the position of the files on the device. This is the same as using /S with no code.
SIZ	Based on file size (in blocks). Files that are the same size are sorted alphabetically by file name and file type. Files are sorted from smallest to largest unless you also use /R.
TYP	Alphabetical by file type. Files that have the same file type are sorted alphabetically by file name.

The following examples illustrate the /S option:

```
*DU0:/S:DAT RET
 4-Feb-91
BUILD .MAC   100  06-Sep-90      SYSMAC.MAC    41  19-Nov-90
DATE  .TXT     3  06-Sep-90      TT    .SYS     2  19-Nov-90
MYPROG.MAC    36P 12-Oct-90      VTMAC .MAC     7  19-Nov-90
RFUNCT.MAC     4  19-Nov-90      TM    .MAC    25  27-Nov-90
RT11SB.SYS    67  19-Nov-90      SWAP  .SYS    25  05-Dec-90
 10 Files, 306 Blocks
 180 Free blocks

*DU0:/S:NAM RET
 4-Feb-91
BUILD .MAC   100  06-Sep-90      SWAP  .SYS    25  05-Dec-90
DATE  .TXT     3  06-Sep-90      SYSMAC.MAC    41  19-Nov-90
MYPROG.MAC    36P 12-Oct-90      TM    .MAC    25  27-Nov-90
RFUNCT.SYS     4  19-Nov-90      TT    .SYS     2  19-Nov-90
RT11SB.SYS    67  19-Nov-90      VTMAC .MAC     7  19-Nov-90
 10 Files, 306 Blocks
 180 Free Blocks

*DU0:/S:POS RET
 4-Feb-91
RT11SB.SYS    67  19-Nov-90       BUILD .MAC   100  06-Sep-90
DATE  .TXT     3  06-Sep-90       SYSMAC.MAC    41  19-Nov-90
MYPROG.MAC    36P 12-Oct-90       TM    .MAC    25  27-Nov-90
SWAP  .SYS    25  05-Dec-90       VTMAC .MAC     7  19-Nov-90
RFUNCT.SYS     4  19-Nov-90       TT    .SYS     2  19-Nov-90
 10 Files, 306 Blocks
 180 Free blocks

*DU0:/S:SIZ RET
 4-Jan-91
TT    .SYS     2  19-Nov-90      TM    .MAC    25  27-Nov-90
DATE  .TXT     3  06-Sep-90      MYPROG.MAC    36P 12-Oct-90
RFUNCT.SYS     4  19-Nov-90      SYSMAC.MAC    41  19-Nov-90
VTMAC .MAC     7  19-Nov-90      RT11SB.SYS    67  19-Nov-90
SWAP  .SYS    25  05-Dec-90      BUILD .MAC   100  06-Sep-90
 10 Files, 306 Blocks
 180 Free blocks

*DU0:/S:TYP RET
 14-Dec-90
BUILD .MAC   100  06-Sep-90      DATE  .TXT     3  06-Sep-90
MYPROG.MAC    36P 12-Oct-90      RFUNCT.SYS     4  19-Nov-90
SYSMAC.MAC    41  19-Nov-90      RT11SB.SYS    67  19-Nov-90
TM    .MAC    25  27-Nov-90      SWAP  .SYS    25  05-Dec-90
VTMAC .MAC     7  19-Nov-90      TT    .SYS     2  19-Nov-90
 10 Files, 306 Blocks
 180 Free blocks
```

DIR Option Descriptions

Protection Option (/T)

The /T option includes in the directory listing only those files on the volume you specify that are protected against deletion. A letter P next to the block size number in the file's directory entry indicates that the file is protected. The following command lists only those files on DK that are protected:

```
*DK:/S:SIZ/R/T RET
 5-Jan-91
BUILD .MAC   100P 06-Sep-90      TM    .MAC    25P 27-Nov-90
RT11SB        67P 19-Nov-90      VTMAC .MAC     7P 19-Nov-90
SYSMAC.MAC    41P 19-Nov-90      RFUNCT.SYS     4P 19-Nov-90
MYPROG.MAC    36P 12-Oct-90      DX    .SYS     3P 06-Sep-90
SWAP  .SYS    25P 05-Dec-90      TT    .SYS 2P 19-Nov-90
 10 Files, 306 Blocks
 5584 Free blocks
```

No Protection Option (/U)

The /U option includes in the directory listing only those files on the volume you specify that are not protected against deletion. Files that are not protected do not have a P in the file's directory entry. The following command lists only those files on DK that are not protected:

```
*/S:SIZ/R/U RET
 14-Dec-90
COUNT .MAC   100 06-Sep-90      SBT   .TXT    25 27-Nov-90
ASCII .MAC    67 19-Nov-90      MAIL  .MAI     7 19-Nov-90
SUBONE.MAC    41 19-Nov-90      SQRT  .FOR     4 19-Nov-90
MYPROG.MAC    36 12-Oct-90      DX    .SYS     3 06-Sep-90
 8 Files, 283 Blocks
 325 Free blocks
```

Volume ID Option (/V[:ONL])

The /V option displays the volume identification and owner name as part of the directory listing header. The optional argument, :ONL, displays only the volume ID and owner name. You can combine /V with any other option.

The following example uses the /V option:

```
*DU:/V RET
 14-Jan-91
 Volume ID: BACKUP2
 Owner    : Marcy
SWAP  .SYS    25P 19-Nov-90      RT11SB.SYS    67P 19-Nov-90
RT11FB.SYS    80P 19-Nov-90      LETTER.TXT    64P 19-Nov-90
TT    .SYS     2P 19-Nov-90      MEMO2 .TXT     3P 19-Nov-90
MEMO1 .TXT     3P 19-Nov-90      DX    .SYS     3P 19-Nov-90
DATE  .TXT     4P 19-Nov-90      RF    .SYS     3P 19-Nov-90
RK    .SYS     3P 19-Nov-90      DL    .SYS     4P 19-Nov-90
 12 Files, 271 Blocks
 215 Free blocks
```

The next example uses the :ONL argument:

```
*DU0:/V:ONL RET
Volume ID: RT11 V5.6
Owner    : Donna
```

DIR Option Summary

The DIR options enable you to perform many kinds of directory operations. Table 5–1 summarizes these operations. The section following the table alphabetically lists and describes each option.

Table 5–1: DIR Options

Option	Description
/A	Lists the directory of the volume you specify in alphabetical order by file name and type (this is the same as /S:NAM).
/B	Lists the directory of the volume you specify, including file names and types, creation dates, starting block numbers, and the number of blocks in each file. For magtape, the starting block number is the file sequence number. Note that DIR lists block numbers in decimal, unless you use the /O option.
/C[:value]	Lists the directory in the number of columns specified by *value*, which is an integer in the range 1–9. The default value is two columns for normal listings and five columns for abbreviated listings.
/D[:date]	Lists a directory containing only those files having the date you specify. If you do not supply a date, DIR uses the system's current date.
/E	Adds unused spaces and their sizes to the listing of the volume directory.
/F	Displays a five-column, short directory (file names and types only) of the volume you specify.
/G	Lists the file you specify and all files that follow it in the directory. This option does not list any files that precede the file you specify.
/J[:date]	Displays a directory of the files created on or after the date you specify. If you do not supply a date, DIR uses the system's current date.
/K[:date]	Displays a directory of files created before the date you specify. If you do not supply a date, DIR uses the system's current date.
/L	Lists the directory of the volume you specify, including the number of files, their dates, and the number of blocks each file occupies. (This is the default operation.)
/M	Lists a directory of unused areas of the volume you specify.
/N	Lists a summary of the device directory.
/O	Similar to /L but lists the sizes and block numbers of the files in octal.
/P	Displays a directory of the volume you specify, excluding the files you list.
/Q	Lists a directory of the volume you specify, listing the file names and types, sizes, creation dates, and starting block numbers of files that have been deleted and whose file name information has not been destroyed.

DIR Option Summary

Table 5–1 (Cont.): DIR Options

Option	Description
/R	Lists the files in the reverse order of the sort specified with /A or /S.
/S[:category]	Lists the directory of the volume you specify in the order you specify; category indicates the order in which DIR sorts the listing (category can be DAT, NAM, POS, SIZ, or TYP).
/T	Lists a directory of all files on the volume you specify that are protected against deletion.
/U	Lists a directory of all files on the volume you specify that are not protected against deletion.
/V[:ONL]	Lists the volume ID and owner name as part of the directory listing header. If you specify /V:ONL, DIR lists only the volume ID and owner name.

DCL Equivalents of DIR Utility Operations

Table 5–2 lists the DCL DIRECTORY command options that are equivalent to DIR utility operations.

The first part of the table lists that part of the DIR command syntax that is equivalent to a DIRECTORY option. The rest of the table alphabetically lists all the DIR options having DCL equivalents.

Table 5–2: DCL Equivalents of DIR Utility Operations

DIR Utility Syntax/Option	DIRECTORY Command Option
filespec=	/OUTPUT:filespec
filespec[size]=	/ALLOCATE:size
LP:=	/PRINTER
TT:= (default)	/TERMINAL (default)
/A	/ALPHABETIZE
/B	/BLOCKS (disks)
/C:value	/POSITION (magtapes) /COLUMNS:value
/D	/NEWFILES
/D[:date]	/DATE[:date]
/E	/FULL
/F	/FAST /BRIEF
/G	/BEGIN
/J[:date]	/SINCE[:date]
/K[:date]	/BEFORE[:date]
/M	/FREE
/N	/SUMMARY
/O	/OCTAL
/P	/EXCLUDE
/Q	/DELETED
/R	/REVERSE
/S[:category]	/SORT[:category] /ORDER[:category]

Table 5–2 (Cont.): DCL Equivalents of DIR Utility Operations

DIR Utility Syntax/Option	DIRECTORY Command Option
/T	/PROTECTION
/U	/NOPROTECTION
/V[:ONL]	/VOLUMEID[:ONLY]

Chapter 6
Dump Utility (DUMP)

The DUMP Utility (DUMP) translates the binary data in all or part of a file or volume into formatted octal words and/or bytes, ASCII characters, and/or Radix–50 characters. DUMP displays this translation in a listing on either the terminal or printer, or writes the listing to a file. For this reason, DUMP is useful for examining the contents of directories, files, and volumes.

For a listing of binary, octal, decimal, and hexadecimal equivalents, see the table listing the DEC Multinational Character Set in the *PDP–11 MACRO–11 Language Reference Manual*. For tables listing both the ASCII and the Radix–50 character sets, see the *RT–11 Quick Reference Manual*.

Calling and Terminating DUMP

You can call the DUMP utility program from the system device by issuing either one of the following two DCL commands at the monitor dot (.) prompt:

```
.DUMP[/options] filespec RET
.R DUMP RET
```

The first command example shows the format for the DCL DUMP command, while the second command runs the DUMP program. The *RT–11 Commands Manual* explains how to use the DCL DUMP command with its options. This chapter explains how to interact with the DUMP utility when you have issued the command R DUMP. Table 6–2 lists the equivalents between the CSI DUMP operations and the DCL DUMP operations.

When you issue the command R DUMP, the DUMP program prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. If you respond to the asterisk by pressing RETURN, DUMP displays its current version number.

You can press CTRL/C to halt DUMP and return control to the monitor when DUMP is waiting for input from the console terminal. You must press CTRL/C twice to abort DUMP at any other time.

Command-Line Syntax

The following CSI command-line syntax presumes you are at the asterisk prompt, having already issued the command to run the DUMP utility (as explained in the previous section):

[outfile[size]=]infile/options

where:

outfile is the optional output file containing the listing that the DUMP utility produces:

- The default listing contains the contents of the specified input file in octal words along with ASCII characters. See Table 6–1 for other forms of DUMP listings.
- If you do not specify an output file, the listing prints on the printer.
- If you specify TT: as the output file, the listing is displayed on the terminal screen.
- The default file specification for the output file is DK:DUMP.DMP. If you do not specify an output device, DUMP uses DK. If you do not specify an output file name, DUMP uses DUMP, and if you do not specify a file type, DUMP uses DMP.

[size] is an optional decimal number that reserves space for the output file on the device. The value of the number is the number of blocks of space to allocate. The meaningful range for this value is from 1 to 65527. A value of -1 is a special case that creates the largest file possible on a device.

Note: Although this is an optional qualifier, the square brackets are a necessary part of the qualifier and are not optional.

infile can be either a file specification or a device specification, depending on whether you want the DUMP utility to examine the contents of a file or a volume. If you want to examine the directory of a volume, you would specify a device with the directory block numbers.

See Chapter 1 for a detailed explanation of the CSI and the equivalent CCL (Concise Command Language) command-line syntax.

DUMP Options

Table 6–1 summarizes the CSI options that are valid for DUMP.

Table 6–1: DUMP Options

Option	Function
/A	Displays the ASCII equivalent of each octal word or byte that is dumped.
/B	Displays the contents of individual bytes in octal.
/E:n	Ends output at block n, where n is an octal block number, unless you make it a decimal number by including a period after the number.
/G	Ignores input errors that occur during a dump operation.
/N	Suppresses ASCII output. ASCII characters are always dumped unless you specify /N.
/O:n	Displays only block n, where n is an octal block number, unless you make it a decimal number by including a period after the number. With this option, you can dump only one block for each command line.
/S:n	Starts output with block n, where n is an octal block number, unless you make it a decimal number by including a period after the number. For random-access devices, n may not be greater than the number of blocks in the file.
/T	Defines a magtape as a device that is not RT–11 file-structured.
/W	Displays words, in octal; words are always dumped unless you specify /B.
/X	Displays Radix–50 characters. An unused reserved Radix–50 character is displayed as an ASCII slash (/).

NOTES

The first block of any file or device is block 0.

If you are dumping a file, the block numbers you specify are relative to the beginning of that file. Whereas, if you are dumping a device, the block numbers are the absolute (physical) block numbers on that device.

RT–11 measures blocks as 512 bytes. However, with FSM.MAC there are some 80-byte blocks. DUMP indicates this. Also, with BUP.SAV, most data has a blocking-factor of 4096 bytes for each block. DUMP indicates this.

DUMP does not print data from track 0 of RX01/RX02 diskettes.

Operations With Magtape

DUMP handles operations that involve magtape differently from operations involving random-access devices.

If you dump an RT–11 file-structured tape and specify only a device name in the input specification, DUMP reads only as far as the logical end-of-tape. Logical end-of-tape is indicated by an end-of-file label followed by two tape marks. For non-file-structured tape, logical end-of-tape is indicated by two consecutive tape marks. For magtape dumps, tape mark messages appear in the output listing as DUMP encounters them on the tape.

If you use /S:n with magtape, n can be any positive value. However, an error can occur if n is greater than the number of blocks written on the tape. For example, if a tape has 100 written blocks and n is 110, an error can occur if DUMP accesses past the 100th block. If you specify /E:n, DUMP reads the tape from its starting position (block 0, unless you specify otherwise) to block n or to logical end-of-tape, whichever comes first.

How to Interpret a DUMP Listing

To understand how DUMP translates binary code, look at the following one-sentence contents of the file FOX.TXT:

```
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.
```

The next two example file listings are the first 64 bytes of two different dumps of the preceding file. The 448 bytes of the two dump listings that are not shown list zeros in the rest of the 512 bytes of each file to show that they contain no information—the smallest unit of information RT–11 deals with on a disk is 1 block (512 bytes). When you create a file with KED/KEX, the editor allocates a minimum of 1 block for your file, even if it contains only a few bytes of information.

First Listing

```
FOX.TXT
BLOCK NUMBER  000000
000/ 044124 020105 052521 041511 020113 051102 053517 020116 *THE QUICK BROWN *
020/ 047506 020130 052512 050115 042105 047440 042526 020122 *FOX JUMPED OVER *
040/ 044124 020105 040514 054532 042040 043517 000056 000000 *THE LAZY DOG....*
060/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
```

The first listing is the default listing, without options. Notice the following in this listing:

- The first line of the listing contains the input-file specification.
- The second line specifies the input-file block number at which the listing starts.
- The left column of numbers with slashes is the *octal* byte offset from the beginning of the block. Each row across represents 16 bytes or 8 words of binary information; the $17th_{10}$ byte is at offset 20, the 33rd byte is at offset 40 and so forth.
- The eight columns following the byte offsets contain eight words in octal code.
- The ASCII equivalent of the eight words is displayed in the column to the right of the octal words.

Second Listing

```
DK:FOX.TXT
BLOCK NUMBER  000000
000/ 044124  020105  052521  041511  020113  051102  053517  020116
     124 110 105 040 121 125 111 103 113 040 102 122 117 127 116 040
     T   H   E       Q   U   I   C   K       B   R   O   W   N
020/ 047506  020130  052512  050115  042105  047440  042526  020122
     106 117 130 040 112 125 115 120 105 104 040 117 126 105 122 040
     F   O   X       J   U   M   P   E   D       O   V   E   R
040/ 044124  020105  040514  054532  042040  043517  000056  000000
     124 110 105 040 114 101 132 131 040 104 117 107 056 000 000 000
     T   H   E       L   A   Z   Y       D   O   G   .   .   .   .
060/ 000000  000000  000000  000000  000000  000000  000000  000000
     000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
     .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
```

The second listing includes the two options /WORDS (specifying octal words) and /BYTES (specifying octal bytes). If you do not include the /WORDS option along with the /BYTES option, the listing will not contain words in octal code.

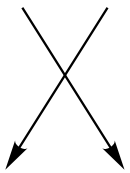
The *RT–11 Quick Reference Manual* has a reference section table listing the left/right byte equivalents for each of the octal numbers from 000 to 377.

The ASCII equivalent of each byte is placed below that byte.

Note the dots in the listing. DUMP uses a dot to represent not only a period but also nonprinting codes, such as those for control characters.

Note also the relationship of the bytes to the words. For example, the first octal word is 044124. That word is divided into a left byte represented by the octal number 124 and a right byte represented by the octal number 110. However, the bytes are displayed in address order; the low-order byte of each word is displayed before the high-order byte. See the following diagram:

```
01 00|1  00|0 01| 010 |100        word in binary
   4     4    1    2    4         represented in octal
01 001 000  |  01 010 100         two-byte components
```



```
01 010 100  |  01 001 000         two bytes in address order
 1  2   4       1  1   0         represented in octal
```

How to Interpret a DUMP of a Directory

One reason for examining volumes is to check the information stored in directories. To understand how to interpret a dump listing of a directory, note the following directory of an RX50 diskette:

```
 13-Feb-91
MEMO1 .TXT     1  13-Feb-91      MEMO2 .TXT     6  13-Feb-91
 2 Files, 7 Blocks
 779 Free blocks
```

The preceding directory listing contains two files. If you examine that directory with the command DUMP/NOASCII/RAD50/ONLY:6, you get the following directory listing:

```
DZ:/N/X/O:6
BLOCK NUMBER  000006
000/ 000004 000000 000001 000000 000016 002000 051025 061230
       D             A             N     YX    MEM    O1
020/ 100324 000001 000000 004663 002000 051025 061300 100324
     TXT      A           AVC     YX    MEM    O2     TXT
040/ 000006 000000 004663 001000 000325 063471 023364 001413
       F           AVC     L2     EM    PTY    FIL     SS
060/ 000000 004663 004000 000000 000000 000000 000000 000000
            AVC    AKH
100/ 000000 000000 000000 000000 000000 000000 000000 000000

120/ 000000 000000 000000 000000 000000 000000 000000 000000
```

Note that only the first 96 bytes of the 512-byte block of the dump listing are shown in the example. Since the listing is of a directory containing only two files, the rest of the listing is of unused bytes. Note also the input file specification at the start of directory dump listing is DU:/N/X/O:6:

DU: Is the device containing the volume with the directory to be examined.

/N Specifies that ASCII output be suppressed. Since ASCII binary code is not used to store information in RT–11 directories, ASCII translations of directory information would produce useless information.

/X Specifies Radix–50 output since RT–11 uses Radix–50 code to store information in directories. This is a code that is more compact than ASCII and can store three characters in a binary word (rather than two). In the listing, the letters and numbers beneath the octal words are the Radix–50 equivalents of those words. If you look carefully at the Radix–50 equivalents, you can see (in groups of two and three alphanumeric characters) the names of the files listed in the preceding directory.

/O:6 Specifies the listing containing only the information in block 6. RT–11 directories on random-access devices always begin in block 6. So, if you want a dump of a directory on a random-access device, begin with block 6; that is, specify /S:6 (for *start at block 6*).

In the example, the option /O (for /ONLY) is the letter *O*. And only a listing of block 6 is requested.

Example Commands and Listings

This section includes sample DUMP commands and the listings they produce.

1. The following command string directs DUMP to print, in words, information contained in block 1 of the file DMPX.SAV stored on device DK:

```
*DMPX.SAV/O:1

DMPX.SAV/O:1
BLOCK NUMBER 000001
000/ 000000 042062 000000 000000 000000 000000 000000 000000 *..2D............*
020/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
040/ 000000 000000 001002 000000 000000 003356 001010 001104 *..........n...D.*
060/ 000014 000400 004001 000000 000000 000000 000000 000000 *................*
100/ 000000 000000 045504 043072 046111 030505 044456 046523 *....DK:FILE1.ISM*
120/ 000000 046061 000000 000000 000000 000000 000000 000000 *..1L............*
140/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
160/ 000000 000000 000000 000000 000000 001356 002000 001234 *..........n.....*
200/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
360/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
400/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
420/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
760/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
```

2. The next command dumps block 1 of file PIP.SAV on the terminal. The /N option suppresses ASCII output:

```
*TT:=PIP.SAV/N/O:1

SY:PIP.SAV/N/O:1
BLOCK NUMBER  000001
000/ 060502 010046 010146 010246 000422 062701 001100 012102
020/ 022512 001406 012100 005046 011146 010246 104217 103405
040/ 012602 012601 012600 011505 000205 104376 175400 012767
060/ 011501 177724 016701 000012 005021 020167 000006 103774
100/ 000743 005562 015260 005562 000006 002112 005562 000013
120/ 003147 014100 000022 000211 014100 000023 000423 014100
140/ 000025 000470 004537 001002 000006 006456 004537 001002
160/ 000014 005764 004537 001002 000022 014102 004537 001002
200/ 000030 014102 004537 001002 000036 014120 000000 000000
220/ 000000 000000 000000 000000 000000 000000 000000 000000
```

```
240/ 000000 000000 000000 000000 000000 000000 000000 000000
260/ 000000 000000 000000 000000 000000 000000 000000 000000
300/ 000000 000000 000000 001000 015260 000000 000000 000000
320/ 000000 000000 000000 000000 000000 000000 000000 000000
340/ 000000 000000 000000 000000 000000 000000 000000 000000
360/ 000000 000000 000000 000000 000000 000000 000000 000000
400/ 000000 000000 000000 000000 000000 000000 000000 000000
420/ 000000 000000 002056 002063 003452 000000 005020 000000
440/ 000000 000000 000000 000000 000000 000000 000000 000000
460/ 000000 000000 000000 000000 000000 000000 000000 000000
500/ 000000 000000 000000 000000 000000 000000 000000 000000
520/ 000000 000000 000000 000000 000000 000000 000000 000000
540/ 000000 000000 000000 000000 000000 000000 000000 000000
560/ 000000 000000 000000 000000 000000 000000 000000 000000
600/ 000000 000000 000000 000000 000000 000000 000000 000000
620/ 000000 000000 000000 000000 000000 000000 000000 000000
640/ 000000 000000 000000 000000 000000 000000 000000 000000
660/ 000000 000000 000000 000000 000000 000000 000000 000000
700/ 000000 000000 000000 000000 000000 000000 000000 000000
720/ 000000 000000 000000 000000 000000 000000 000000 000000
740/ 000000 000000 000000 000000 000000 000000 000000 000000
760/ 000000 000000 000000 000000 000000 000000 000000 000000
```

3. The following command dumps block 0 of SAMPLE.KED in bytes into the file CHECK.DMP on device DK. ASCII equivalents appear underneath each byte:

```
*CHECK=SAMPLE.KED/B/O:0

SY:SAMPLE.KED/B/O:0
BLOCK NUMBER  000000
000/ 040 123 141 155 160 154 145 040 113 145 171 160 141 144 040 105
         S   a   m   p   l   e       K   e   y   p   a   d       E
020/ 144 151 164 151 156 147 040 123 145 163 163 151 157 156 040 055
     d   i   t   i   n   g       S   e   s   s   i   o   n       -
040/ 040 057 057 104 101 124 105 057 057 015 012 015 012 040 040 040
         /   /   D   A   T   E   /   /   .   .   .   .
060/ 040 040 124 150 151 163 040 146 151 154 145 040 150 141 163 040
             T   h   i   s       f   i   l   e       h   a   s
100/ 142 145 145 156 040 144 145 163 151 147 156 145 144 040 145 163
     b   e   e   n       d   e   s   i   g   n   e   d       e   s
120/ 160 145 143 151 141 154 154 171 040 146 157 162 040 164 150 145
     p   e   c   i   a   l   l   y       f   o   r       t   h   e
140/ 040 163 141 155 160 154 145 040 145 144 151 164 151 156 147 040
         s   a   m   p   l   e       e   d   i   t   i   n   g
160/ 163 145 163 163 151 157 156 015 012 164 150 141 164 040 151 163
     s   e   s   s   i   o   n   .   .   t   h   a   t       i   s
200/ 040 144 145 163 143 162 151 142 145 144 040 151 156 040 103 150
         d   e   s   c   r   i   b   e   d       i   n       C   h
220/ 141 160 164 145 162 040 061 040 157 146 040 164 150 145 040 120
     a   p   t   e   r       1       o   f       t   h   e       P
240/ 104 120 055 061 061 040 113 145 171 160 141 144 040 105 144 151
     D   P   -   1   1       K   e   y   p   a   d       E   d   i
260/ 164 157 162 040 125 163 145 162 047 163 040 107 165 151 144 145
     t   o   r       U   s   e   r   '   s       G   u   i   d   e
300/ 056 015 012 015 012 101 146 164 145 162 040 171 157 165 040 150
     .   .   .   .   .   A   f   t   e   r       y   o   u       h
320/ 141 166 145 040 143 157 155 160 154 145 164 145 144 040 164 150
```

```
     a   v   e       c   o   m   p   l   e   t   e   d       t   h
340/ 145 040 163 141 155 160 154 145 040 163 145 163 163 151 157 156
     e       s   a   m   p   l   e       s   e   s   s   i   o   n
360/ 040 144 145 163 143 162 151 142 145 144 040 151 156 040 103 150
         d   e   s   c   r   i   b   e   d       i   n       C   h
400/ 141 160 164 145 162 040 061 054 015 012 171 157 165 040 155 141
     a   p   t   e   r       1   ,   .   .   y   o   u       m   a
420/ 171 040 165 163 145 040 164 150 151 163 040 146 151 154 145 040
     y       u   s   e       t   h   i   s       f   i   l   e
440/ 164 157 040 160 162 141 143 164 151 143 145 040 157 164 150 145
     t   o       p   r   a   c   t   i   c   e       o   t   h   e
460/ 162 040 153 145 171 160 141 144 040 145 144 151 164 157 162 040
     r       k   e   y   p   a   d       e   d   i   t   o   r
500/ 146 165 156 143 164 151 157 156 163 040 141 156 144 040 015 012
     f   u   n   c   t   i   o   n   s       a   n   d       .   .
520/ 143 157 155 155 141 156 144 163 054 040 151 146 040 171 157 165
     c   o   m   m   a   n   d   s   ,       i   f       y   o   u
540/ 040 154 151 153 145 056 015 012 015 012 101 102 117 125 124 040
         l   i   k   e   .   .   .   .   .   A   B   O   U   T
560/ 124 110 105 040 123 101 115 120 114 105 040 123 105 123 123 111
     T   H   E       S   A   M   P   L   E       S   E   S   S   I
600/ 117 116 015 012 015 012 131 157 165 162 040 147 145 156 145 162
     O   N   .   .   .   .   Y   o   u   r       g   e   n   e   r
620/ 141 154 040 164 141 163 153 040 146 157 162 040 164 150 145 040
     a   l       t   a   s   k       f   o   r       t   h   e
640/ 163 141 155 160 154 145 040 163 145 163 163 151 157 156 040 151
     s   a   m   p   l   e       s   e   s   s   i   o   n       i
660/ 163 040 164 157 040 151 156 163 145 162 164 040 164 150 145 040
     s       t   o       i   n   s   e   r   t       t   h   e
700/ 144 141 164 145 040 171 157 165 040 015 012 142 145 147 151 156
     d   a   t   e       y   o   u       .   .   b   e   g   i   n
720/ 040 167 157 162 153 151 156 147 040 167 151 164 150 040 164 150
         w   o   r   k   i   n   g       w   i   t   h       t   h
740/ 145 040 153 145 171 160 141 144 040 145 144 151 164 157 162 040
     e       k   e   y   p   a   d       e   d   i   t   o   r
760/ 151 156 164 157 040 171 157 165 162 040 157 167 156 040 143 157
     i   n   t   o       y   o   u   r       o   w   n       c   o
```

4. The final example command places the contents of block 6 (the directory) of device DU1 into the file DUMP.DMP on logical device LD1. The output is in octal words with Radix–50 equivalents below each word:

```
*LD1:=DU1:/N/X/O:6

DU1:/N/X/O:6
BLOCK NUMBER  000006
000/ 000020 000002 000004 000000 000046 002000 075131 062000
       P      B      D              8     YX    SWA    P
020/ 075273 000031 000000 027147 002000 071677 142302 075273
     SYS      Y           GP9     YX    RT1    1SJ    SYS
040/ 000103 000000 027147 002000 071677 141262 075273 000120
      A$           GP9     XY    RT1    1FB    SYS     B
060/ 000000 027147 002000 071677 141034 075273 000100 000000
            GP9     YX    RT1    1BL    SYS     AX
100/ 027147 002000 100040 000000 075273 000002 000000 027147
     GP9     YX    TT            SYS      B           GP9
120/ 002000 016040 000000 075273 000003 000000 027147 002000
```

```
      YX    DT            SYS      C           GP9     YX
140/ 015600 000000 075273 000003 000000 027147 002000 016300
     DP            SYS      C           GP9     YX    DX
160/ 000000 075273 000003 000000 027147 002000 016350 000000
            SYS      C           GP9     YX    DY
200/ 075273 000004 000000 027147 002000 070560 000000 075273
     SYS      D           GP9     YX    RF           SYS
220/ 000003 000000 027147 002000 071070 000000 075273 000003
       C           GP9     YX    RK           SYS      C
240/ 000000 027147 002000 015340 000000 075273 000004 000000
            GP9     YX    DL           SYS      D
260/ 027147 002000 015410 000000 075273 000005 000000 027147
     GP9     YX    DM           SYS      E           GP9
300/ 002000 015770 000000 075273 000003 000000 027147 002000
      YX    DS            SYS      C           GP9     YX
320/ 014640 000000 075273 000005 000000 027147 002000 046600
     DD            SYS      E           GP9     YX    LP
340/ 000000 075273 000002 000000 027147 002000 046770 000000
            SYS      B           GP9     YX    LS
360/ 075273 000002 000000 027147 002000 012620 000000 075273
     SYS      B           GP9     YX    CR           SYS
400/ 000003 000000 027147 002000 052070 000000 075273  000011
       C           GP9     YX    MS           SYS       I
420/ 000000 027547 002000 052150 014400 075273 000003  000000
            GWO     YX    MTH    D      SYS      C
440/ 027147 002000 015173 052177 012445 00011 000000 027547
     GP9     YX    DIS    MT1    COM      I          GWO
460/ 002000 051520 014400 075273 000004 000000 027147 002000
      YX    MMH    D      SYS      D           GP9     YX
500/ 015173 052200 012445 000010 000000 027547 002000 052100
     DIS    MT2    COM      H           GWO     YX    MSH
520/ 014400 075273 000004 000000 027147 002000 054540 000000
     D      SYS      D           GP9     YX    NL
540/ 075273 000002 000000 027147 002000 062170 000000 075273
     SYS      B           GP9     YX    PC           SYS
560/ 000002 000000 027147 002000 062240 000000 075273 000003
       B           GP9     YX    PD           SYS      C
600/ 000000 027147 002000 012740 000000 075273 000005 000000
            GP9     YX    CT           SYS      E
620/ 027147 002000 006250 000000 075273 000007 000000 027147
     GP9     YX    BA           SYS      G           GP9
640/ 002000 016130 000000 073376 000051 000000 027147 002000
      YX    DUP           SAV     AA           GP9     YX
660/ 023752 050574 073376 000023 000000 027147 002000 070533
     FOR    MAT    SAV      S           GP9     YX    RES
700/ 060223 073376 000017 000000 027147 002000 015172 000000
     ORC    SAV      O           GPO     YX    DIR
720/ 073376 000021 000000 027147 002000 075273 050553 074324
     SAV      Q           GPO     YX    SYS    MAC    SML
740/ 000052 000000 027147 002000 017751 076400 073376 000023
      AB          GP9     YX    EDI    T      SAV      S
760/ 000000 027147 002000 042614 000000 073376 000073 000000
            GP9     YX    KED          SAV     AS
```

DCL Equivalents of DUMP Utility Operations

Table 6–2 lists the options of the DCL DUMP command that are equivalent to CSI DUMP utility operations. The first part of the table lists those DCL options that are equivalent to utility commands without options. The second part of the table lists DCL options that are equivalent to CSI options. The CSI options are listed in alphabetical order.

The value for the *n* arguments to both the DCL and the CSI options is octal by default. But the value for the *size* argument is decimal by default.

Table 6–2: DCL Equivalents of DUMP Utility Operations

CSI Command/Option	DCL Option
infile	/ASCII/PRINTER (the default)
outfile=infile	/OUTPUT:file
outfile[size]=infile	/OUTPUT:file/ALLOCATE:size
TT:=infile	/TERMINAL
/B	/BYTES
/E:n	/END:n
/G	/IGNORE
/N	/NOASCII
/O:n	/ONLY:n
/S:n	/START:n
/T	/FOREIGN
/W	/WORDS
/X	/RAD50

Chapter 7
Device Utility (DUP)

The Device Utility (DUP) is a device maintenance program that:

- Boots RT–11 file-structured volumes.
- Consolidates files and free space on disks or diskettes.
- Prepares bootable volumes.
- Copies volumes in image-mode.
- Creates files on file-structured RT–11 volumes (disks, single- and double-density diskettes, and magtape).
- Displays and/or changes the volume ID and owner's name of a volume.
- Extends files on certain file-structured volumes (disks, and single- and double-density diskettes).
- Initializes or restores the directory of file-structured volumes.
- Scans for bad blocks and replaces or covers them on a volume.

DUP does not operate on non-file-structured devices (printer, terminal).

Calling and Terminating DUP

To call DUP from the system device, respond to the dot prompt (.) displayed by the keyboard monitor by entering:

```
.R DUP RET
```

The Command String Interpreter (CSI) displays an asterisk (*) at the left margin of the terminal and waits for you to issue a command string. If you press only RETURN at this point, DUP displays its current version number and prompts you again for a command string.

You can type CTRL/C to terminate DUP and return control to the monitor when DUP is waiting for input from the terminal. However, you must press CTRL/C twice to abort DUP at any other time. Note that the /S, /T, and /C operations, each lock out the CTRL/C command until the operation completes; these three operations cannot be interrupted with CTRL/C. To restart DUP, enter R DUP or REENTER in response to the monitor's dot prompt.

To call DUP through KMON commands, see the DUP action-option descriptions in the DUP Option Summary. The KMON command equivalent(s) for each option that has them is listed in the action option descriptions.

Command-Line Syntax

The following CSI command-line syntax assumes you are at the asterisk prompt, having already issued the command to run the DUP utility (as explained in the preceding section):

outfile=infile/options

You can have only one input file specification and one output file specification and the syntax of the command line varies slightly depending on the option(s) used. Each DUP option description begins with the syntax needed for that option. See Chapter 1 for a more detailed general explanation of the CSI and the equivalent CCL (Concise Command Language) command-line syntax.

Two Types of DUP Options

You can use two types of options with DUP: *action* and *mode*. Use action options for creating files, copying devices, scanning for bad blocks, booting a device, and initializing a volume. Use mode options to modify the action options when necessary.

Usually, you can specify only one action option at a time. Table 7–1 lists all the action options with all the mode options that can modify each action. Note that /V can be either an action or a mode option, depending on how you use it.

Table 7–1: DUP Option Combinations

Action	Mode
C	G, W, Y
D	W, Y
I	E, F, G, H, J, W, Y
K	E, F, G, H, R, W
O	Q, W, Y
S	W, X, Y
T	W, Y
U	W, Y
V	W, Y
Z	B, D, H, N, R, V, Y, W

DUP Option Summary

Table 7–2 lists each DUP option and summarizes option functions.

Table 7–2: Summary Descriptions of DUP Options

Option	Function
/B[:RET]	Covers bad blocks. Use with the /Z option to write files with the file type .BAD over any bad blocks DUP finds on the disk to be initialized. Use :RET to retain through initialization all .BAD entries created by a previous /B.
/C	Creates a file on the volume you specify. DUP creates the file in the first available location, unless you specify a starting-block number by using the /G option.
/D	Restores a volume's directory. Use with the /Z option to restore (uninitialize) a volume. Use only if no files have been transferred to the volume since it was initialized.
E:value	Specifies the octal ending-block number for a read operation. Use with the /I and/or /K.
/F	Copies a file to or from a volume; or displays the file names in which bad blocks occur. Use with the /I option either to copy a file to an output volume or to copy a volume to an output file. Use with the /K option to transfer the file name containing the bad block together with the relative block number of the bad block in the file.
/G:value	Specifies the octal starting-block number for a read operation (on an input device) and the octal starting-block number for a write operation (on an output device). Use this option with the /C, /I, and /K options.
/H	Verifies that output equals input; use with the /I, /K, and /B options. CSI command only; not implemented as KMON command option. RT–11 reads each block and, if encountering errors, writes out and rereads the block. Use /H/K or /H/B only with blank media or media you have backed up. Use with /K and /B is valid means for clearing bad blocks caused by soft errors on MSCP class devices.
/I	Copies the image of a disk to another disk or magtape or from magtape to disk. Use with the /G and /E options if you want to specify block numbers.
/J	Ignores (skips) a bad block on the input or output device and displays an error message while proceeding with the copy operation.
/K	Scans a volume for bad blocks and outputs the octal address of the bad blocks to the output device. Use with the /G and /E options if you want to specify block numbers as boundaries for the scan.
/N:value	Sets the number of directory segments you specify, if you do not want the default number; *value* is an integer in the range 1–37 (octal). Use with the /Z option.
/O	Boots the volume or monitor file you specify.
/Q	Boots a volume that is not RT–11 or pre-Version 4 volume of RT–11. Use with /O option.

DUP Option Summary

Table 7–2 (Cont.): Summary Descriptions of DUP Options

Option	Function
/R[:RET]	Replaces bad blocks when you initialize a volume or preserves a volume's replacement table when you image-copy the volume. Not supported with the /I option. Use with the /Z option to scan a device that supports bad block replacement for bad blocks. /R creates a replacement table on the disk for any bad blocks DUP finds. If you use /R:RET with /Z, DUP retains the replacement table that is already on the disk and does not pre-scan the disk for bad blocks.
/S	Compresses a volume onto itself or onto another volume. The output device, if any, must be initialized.
/T:value	Extends an existing file by the number of blocks that *value* specifies.
/U[:dev]	Copies the bootstrap portion of the monitor file to blocks 0 and 2–5 of a volume. The optional argument, *dev,* specifies the target system device, if it is different from the input device.
/V[:ONL]	Displays and/or changes a volume's user ID and owner name. Use /V with the /Z option to place a new user ID and owner name in block 1 of the initialized disk or in the VOL1 header block on magtape. Using /V:ONL with /Z changes only the ID and owner name, and does not initialize the device (not applicable for magtape).
/W	Enables you to change volumes during an operation. Initiates any action-option operation and then pauses to let you change volumes. This option is useful on small, single-disk systems because it lets you replace the system volume with another disk before performing an operation. Use with any action option.
/X	Inhibits automatic booting of the system device when it is compressed. Use with /S.
/Y	Ensures immediate execution of an operation by inhibiting confirmation messages. Use with /C, /I, /O, /S, /T, /U, /V, or /Z.
/Z[:value]	Initializes the directory of the volume you specify. The size of the directory defaults to the standard RT–11 size; use *value* to allocate extra directory words for each entry beyond the default.

Create Option (/C[/G:value])

The /C option creates a file with a specific name, location, and size on the random-access device that you specify. This option creates only a directory entry for a file. It does not store any data in the file.

Syntax

outfile[size]=/C[/G:value][/W][/Y]

where:

outfile[size] specifies the device, file name, and file type of the file to be created. You must specify both the file name and file type of the file to be created.

[size] is a decimal number specifying the size in blocks of the file to be created. Note that the brackets here are part of the command; that is, they do not indicate n is optional. A value of -1 indicates a file of the maximum size available on the volume. If you do not specify this number, DUP creates a one-block file.

/G:value specifies the octal numeric value of the starting block of the file to be created. If you do not use /G:value, DUP creates the file in the first unused area large enough to contain the file. Use a decimal point with *value* (*value.*) to specify a decimal starting-block number.

/W initiates the create operation and then pauses to let you mount the volume on which you want to create a file.

/Y suppresses confirmation messages to ensure immediate execution of the operation.

NOTE

The default numeric value for outfile[size] is decimal while the default numeric value for /G:value is octal.

Usage

- Creates files you can assign as logical disks.
- Covers bad blocks on a disk by creating a file with a file type .BAD to cover the bad area.
- Recovers files you accidentally delete. In this case, first use DIR with the /E and /Q options to list files, tentative files, empty areas, and the sizes of all areas. Then assign a file name to the area that contains the data you lost.
- Saves file space by reserving an area on a disk without performing any input or output operations.

Cautions

- You need adequate contiguous space on a disk. When you use the /C option, make sure that the area in which the file is to be created is empty (using the DIR /E) and /Q options). If there are not enough empty contiguous blocks to hold

the file you want to create, DUP displays the error message *?DUP-F-No room for file DEV:FILNAM.TYP* and does not create the file.

- Your file name must be unique.

  The /C option checks for duplicate file names. If the file name you specify already exists on the device, DUP issues an error message and does not create a second file with the same name.

- Creating a file over a tentative file can cause unpredictable results.

  If you attempt to create a file over a tentative file (one that was opened but never closed) and the foreground is loaded, the system prompts you to confirm the operation. If you enter Y to continue, DUP writes over the tentative file. Be sure that you do not write over a tentative file being used by another job; this will corrupt the file and cause unpredictable results.

Example

The following command creates the file FILE.MAC, consisting of blocks 140_8, 141_8, and 142_8 on device DU1:

```
*DU1:FILE.MAC[3]=/C/G:140
```

File Option (/F)

File option serves two different purposes as a mode option, depending on whether you use it with /I or with /K.

Usage

- Usage with /I/F

  When you use /F with /I, use it either to copy a file from an input volume to an output volume, or to copy an input volume to an output file.

  Note that /I does not copy track 0 of RX01 and RX02 diskettes. If you use a magtape for either the input or output volume, you must specify a file name for the magtape followed by the /F option. Do not include wildcards in either the input or output file specification when you use the /F option.

- Usage with /K/F

  When you use /F with /K, DUP does a bad-block scan and displays:

  — The relative block number (in both octal and decimal) of each bad block within the scanned volume.

  — The word *hard* or *soft* beside each block number to indicate the type of error (either a hardware or a software error).

  — Either the name of the file in which each bad block occurs or the message `< UNUSED >` beside those bad blocks that are not used.

  — The relative block number (in both octal and decimal) of each bad block within the scanned file or UNUSED area.

Examples

1. The following example illustrates a bad-block display:

```
*DY0:/K/F RET
    Block     Type           File          Block

000717     463.  Hard     NUMBER.PAS   000002       2.
000725     469.  Hard     ANTONY.MAC   000005       5.
000732     474.  Hard     CAESAR.MAC   000010       8.
000743     483.  Hard     < UNUSED >   000003       3.
000751     489.  Hard     < UNUSED >   000001       1.
000754     492.  Hard     < UNUSED >   000014      12.
?DUP-W-Bad blocks detected 6.
```

2. The next example shows a bad-block display of a disk that has a bad-block replacement table:

```
*DM1:/K/F RET
    Block          Type            File            Block
003055   1581.    Replaced      MSX   .SYS     000007        7.
003465   1845.    Replaced      DRV   .OBJ     000077       63.
037061  15921.    Replaced      < UNUSED >     010550     4456.
056106  23622.    Replaced      < UNUSED >     027575    12157.
056210  23688.    Replaced      < UNUSED >     027677    12223.
077521  32593.    Replaced      < UNUSED >     051210    21128.
143116  50766.    Replaced      < UNUSED >     043374    18172.
145337  51935.    Replaced      < UNUSED >     045615    19341.
?DUP-W-Bad blocks detected 8.
```

When you use /F with /K, on a disk that supports bad-block replacement, in the column marked *Type*, DUP lists whether the bad block is replaced in the manufacturer's bad-block replacement table or if it is hard or soft.

Image-Mode Copy Option (/I)

The /I option copies block-for-block the image of one device to another, and copies all data from one disk to another without changing the file structure or the location of the files on the device.

Syntax

outdevice: { **filename**}
[/F][/G:value]=indevice[filename]/I[/G:value/E:value][/F][/H][/J][/W][/Y]
{ ***}**

where:

filename specifies the output file to which you are copying the input device, or (when specified with the input device) specifies the input file you are copying to the output device.

You can specify a file name with either the input or the output, but never with both.

If you specify an input file name, you must use the dummy file name * with the output specification.

When you specify a file name, you must also specify the /F option along with the name.

***** (asterisk) specifies a dummy file name. Required when you do not use the /F option with the input specification, and when the output device is not a magtape. You can specify either a file name or an asterisk (*) with the output device, but not both.

/G:value when specified with the output device, specifies the octal starting-block number for the write operation. When specified with the input device, it specifies the octal starting-block number of the read operation.

/E:value specifies the octal ending-block number on the input device for the read operation.

/F specifies that you want to copy a file to an output volume, or that you are copying an entire input volume to an output file.

You must use the /F option when you specify magtape as the input or output device (because you must always specify a file on the magtape). DUP consults internal tables to determine if the device is magtape.

If you use the /F option, the relative sizes of the input and output volumes are ignored and you are not asked to confirm the copy operation.

/H verifies that the input matches the output. That is, you can use the /H option with /I to verify that the input matches the output after an image-mode copy operation.

/J ignores (skips) a block containing an error during an image-mode copy operation and proceeds with the copy operations.

COPY/DEVICE/IGNORE causes any errors returned by a bad block on the input or output device to be ignored. The bad block on the device that returns the error and a corresponding block on the other device are not copied. An error message displays which device (input or output) contains the bad block and the bad block number.

/W initiates the copy operation and then pauses to let you mount the volumes you want to operate on.

/Y suppresses confirmation messages to ensure immediate execution of the operation.

The command string must include an input and an output specification; there is no default device.

Usage

Copies entire image of disks or diskettes.

Copies one disk to another without changing the contents of the disk. That is, /I copies the boot block along with the directory, the files, the file locations, and the file structures.

This copy operation does not change the file structure of a volume. So, you can image copy disks that are not in RT–11 format, if they have no bad blocks. If DUP encounters a bad block on either the input or output volume, it retries the operation and performs the copy one block at a time. If no error message displays, you can assume that the transfer completed correctly.

This operation is applicable for magtape only when copying to or from a random-access volume, such as disk or diskette. Because magtapes are not file-structured, only one disk image fits on a magtape. The data stored on tape is formatted in 512-byte blocks.

Options

Depending on the mode options you combine with the /I option and your input and output specifications, you can:

- Indicate the blocks to be read from the input volume (*=infile/I/G:value /E:value).
- Indicate the starting-block number for the write operation on the output volume (outfile/G:value=infile/I).
- Copy a file to a volume (*=infile/I/F) or a volume to a file (outfile/f=indevice/I). For example, you can copy a diskette to a file on an RL02, or a file on an RL02 to a diskette.
- Preserve the output volume's bad-block replacement table when you are copying between like volumes that support bad-block replacement — RK06, RK07, RL01, or RL02 disks (*=infile/I/R).
- Verify that the output matches the input after a copy operation (*=infile/I/H).
- Inhibit confirmation messages to ensure immediate execution of the operation (*=infile/Y).
- Initiate the copy operation and then mount the volume you want to copy (*=infile/Y).

NOTE

When you use /I in an operation involving magtape, you must specify a file name and follow it with the /F option.

Cautions

- Make sure you do not write over bad blocks on the output device, since this operation can write over bad blocks. Either make sure the output volume contains no bad blocks; or, if the output volume has a bad-block replacement table, use the /R option to preserve this table.
- Copying (in image mode) a file to a volume means block zero of your file is copied to block zero of your output volume. So, it is unwise to image copy files to a diskette since you then lose the directory and boot blocks on the diskette and have no directory record of where on the diskette the file ends.
- Make sure the output volume is the right size for your copy operation.
  - — If one volume is smaller than the other volume, DUP copies only the number of blocks of the smaller volume.
  - — If the input volume is larger than the output volume, DUP copies the entire directory of the input volume, but not all of its files.

    So, when you use the /I option to copy a larger volume to a smaller one, DUP asks for confirmation before copying the volume. So also, if you use

the /G:value and /E:value options, DUP asks you to confirm the copy if the number of blocks to be copied is larger than the area on the output volume defined by the /G:value option and the end of the output volume.

— If the input volume is smaller than the output volume, the extra space on the output volume becomes unavailable, since the directory of the smaller volume is copied. So in this case, DUP also asks for confirmation before copying the volume.

NOTE

The /I option does not copy track 0 of RX01 and RX02 diskettes. However, this restriction has no impact on any copy operations involving RT–11 formatted diskettes.

Examples

The following examples use the /I option. The file name * is not significant; it is a dummy file name required by the Command String Interpreter.

1. The first command copies all the blocks from DU0 to DU1:

```
*DU1:*=DU0:/I [RET]
DU1:/Copy; Are you sure? Y [RET]
```

2. The second command copies blocks $0–500_8$ from DU0 to blocks $501–1000_8$ on DU1:

```
*DU1:*/G:501=DU0:/I/G:0/E:500 [RET]
DU1:/Copy; Are you sure? Y [RET]
```

3. The last command copies the volume on device DU0 to a file named SYSTEM.BAK on DU1:

```
*DU1:SYSTEM.BAK/F=DU0:/I [RET]
DU1:/Copy; Are you sure? Y [RET]
```

Bad-Block Scan Option (/K)

Some mass storage volumes (disks, diskettes, and DECtape II) have bad blocks, or they develop bad blocks as a result of age and use. You can use the /K option to scan a volume, locate bad blocks on it, and display the absolute block numbers of those blocks on the terminal or in a file. You can scan an entire volume or only a section of a volume. A complete scan of a volume takes from one to several minutes depending on the size of the volume.

DUP does not destroy data that is stored on the device; and if DUP finds no bad blocks, it displays an informational message.

Syntax

[outfile=]indevice:/K[/G:value][/E:value][/F][/K][/H][/R][/W]

where:

outfile specifies the output file containing the bad block report. If no bad blocks are found, no file is created.

indevice specifies the volume to be scanned.

/G:value specifies the octal block number of the first block to be scanned. If you specify only a starting-block number, DUP scans from that block to the end of the volume.

/E:value specifies the octal block number of the last block to be scanned.

/F:value displays both the name of the file containing the bad block and the relative block number within the file that is bad.

/H clears bad blocks caused by soft errors on RD31, RD32, RD51, RD52, RD53, and RD54 MSCP class devices. You must do a SYSGEN and request DU bad-block replacement support to use the /H option with RC25, RA60, RA80, and RA82 devices.

The /H option causes RT–11 to read each block and, if it encounters an error, write the block out and then read it again. This forces the controller to clear (make useable) blocks that had previously returned a soft error. /K/H is intended for use primarily with MSCP devices. Use /K/H only with blank media or with media you have backed up. Note that this functionality is available only as a CSI-level command and is not implemented as a KMON command option.

/R (if bad blocks are found), creates a replacement table so that routine operations access good blocks instead of bad ones. This option is valid only for RK06, RK07, RL01, and RL02 disks.

/W initiates the scan operation and then pauses to let you mount the volume you want to scan.

The first block on a device is block 0. While the block numbers used with the CSI /G:value and /E:value options are octal by default, the block numbers used with the KMON /START:value and END:value options are decimal by default.

Verifying Bad-Block Reports

Blocks reported as bad can recover when they are caused by soft, rather than hard errors. Therefore, when you get a bad block report, you should do a second bad-block scan and compare the two reports. Blocks reported as bad on both reports are caused by hard errors and cannot recover. Blocks that are reported as bad on the first report, but not on the second report indicate that a soft error has occurred, and the blocks have recovered.

Examples

1. This command scans the entire diskette in DU1:

   ```
   *DU1:/K RET
   ```

2. The next command scans blocks 100 to 200(decimal) of the diskette in DU1 and sends the bad-block report to DU0:BLOCKS.BAD:

   ```
   *DU0:BLOCKS.BAD=DU1:/K/G:100./E:200. RET
   ```

Boot Option (/O)

The /O option can do either of two boot operations:

- A hardware bootstrap of a specific device containing an RT–11 system.
- A software bootstrap of a specific RT–11 monitor file without using the bootstrap blocks on the device.

Syntax

device:/O[/Q][/W][/Y]

or

[device:]monitor-file/O[/Q][/W][/Y]

where:

device	specifies the device you want to boot. See the Bootable Devices section for a list of the devices you can boot.
monitor-file	specifies the monitor file you want to boot: a different monitor on the same device or a monitor on a different device, if you have included a device specification with the file specification.
/O	specifies the boot operation.
/Q	specifies that you want to boot a volume that has a monitor other than the RT–11 Version 4 or 5 monitor. Note: you must use /Q to boot any version of RT–11 prior to Version 4.
/W	initiates the boot operation and then pauses to let you mount the volume you want to boot.
/Y	suppresses confirmation messages to ensure immediate execution of the operation.

Bootable Devices

Valid supported devices		
DL	DM	DU
DX	DY	RK
VM		

Valid unsupported devices		
DD	DP	DS
DT	DW	DZ
PD	RF	

Whether bootstrapping a specific monitor or a specific device, DUP checks to see if the bootstrap blocks are correctly formatted. If the boot operation you request is invalid, DUP displays an error message and waits for another command.

When you reboot with the /O option by itself, you do not have to reenter the date and time of day with the monitor DATE and TIME commands. However, the clock does lose a few seconds during the reboot.

DUP does not retain the date and time when you use the /Q option.

Examples

1. The first command boots the SB monitor on device DU0:

   ```
   *DU0:RT11SB.SYS/O RET
   ```

2. The second command boots a different monitor, the FB monitor, which is also on device DU0:

   ```
   *DU0:RT11FB.SYS/O RET
   ```

3. The final command boots an RT–11 Version 3B volume.

   ```
   *DY0:/O/Q RET
   RT-11SJ V03B-00B
   ```

Squeeze Option (/S)

/S consolidates all the unused blocks on a volume (disk or diskette) into a single area on the device you specify and consolidates the directory entries on that device. During the consolidation process, DUP moves all the files to the beginning of the specified volume, producing a single, unused area following the group of files.

Because the squeeze operation does not move the bootstrap blocks of a volume, you can still boot the volume. Because it does not move files with BAD file types, you do not reuse bad blocks that may occur on the volume.

Syntax

[outdevice=]indevice/S[/W][/X][/Y]

where:

outdevice specifies where you want the compressed copy of the input volume. The output volume must be an initialized random-access volume.

If you specify an output device, you must also specify an asterisk (*) in place of the file name.

Because /S option does not copy boot blocks, you must copy the boot block in a separate operation to make the output volume bootable (see the DUP /U option).

If you specify an output volume, DUP does not request confirmation before it performs the operation. If you do not specify an output volume, DUP displays the *Are you sure?* message and waits for your response before proceeding. You must press Y RETURN for the command to be executed.

indevice specifies the volume you want to squeeze.

If you specify an output device that is different from the input device, then the input volume remains unchanged. But if you specify the same device for the output device as the input device, or if you omit the output device, then the input volume is changed (that is, the input volume is compressed on itself).

/S specifies the squeeze operation.

/W initiates the squeeze operation and then pauses to let you mount the volume you want to squeeze.

/X inhibits the automatic booting of the system device when it is compressed. Use /X only if you are certain the monitor file will not move. And even then, you should reboot the system when the squeeze operation completes, if the device handlers have moved.

/Y ensures immediate execution of the squeeze operation by inhibiting confirmation messages.

Cautions

- Scan a volume (with /K) before you use /S to check for bad blocks so that, if you find any bad blocks, you can rename files containing those blocks, giving them a

BAD file type, and therefore cause DUP to leave them in place when you execute a /S.

During a squeeze operation, files with a BAD file type are renamed FILE.BAD. DUP inserts files before and after BAD files until the space between the last file it moved and the BAD file is smaller than the next file to be moved.

- Do not attempt to squeeze any volume that a running foreground job is using. Data can be written over a file that the foreground job has open, thereby corrupting the file and possibly causing a system crash.

  If you attempt to squeeze the system device (SY) when a foreground or system job is loaded on that volume, you will get an error message and DUP will ignore the /S operation. You must unload the foreground job before using the /S option.

- If you compress your system volume, make sure the DUP program has the name DUP.SAV. If not, a system failure can occur.

If an error occurs during a squeeze operation, DUP continues the operation, performing it one block at a time. If no error message displays, you can assume that the operation completed correctly.

NOTE

If you perform a compress operation on the system volume, the system automatically reboots when the compress operation is completed (unless you specify /X with/S). This occurs to prevent system crashes that can occur when a system file is moved.

Examples

1. This command compresses the files on the system volume and reboots the system when the compress operation completes:

```
*SY:/S
SY:/Squeeze; Are you sure? Y RET

RT-11XM    V05.5
```

2. This command transfers all the files from device DU1 to device DU0, leaving DU1 unchanged:

```
*DU0:*=DU1:/S RET
```

   The file name * is not significant; it is a dummy file name required by the Command String Interpreter.

Extend Option (/T:value)

The /T option extends the size of a file.

Syntax

filespec=/T:value[/W][/Y]

where:

filespec	specifies the device, file name, and file type of the file to be extended.
n	specifies the number of blocks to add to the file.
/W	initiates the extend operation and then pauses, to let you change volumes.
/Y	ensures immediate execution of the extend operation by inhibiting confirmation messages.

You can extend a file in this manner only if it is followed by an unused area of at least n blocks. Any blocks not required by the extend operation remain in the unused area.

Example

This command extends the file ZYZ.TST on device DU1 by 100 blocks:

```
*DU1:ZYZ.TST=/T:100 RET
```

Bootstrap-Copy Option (/U[:dev])

The /U option copies bootstrap information from monitor and handler files to blocks 0 and 2 through 5 of a random-access volume, permitting you to use that volume as a system volume. A volume has to have the bootstrap information in blocks 0 and 2 through 5 for you to be able to boot it and use it as a system volume.

Syntax

outdevice:*=indevice:monitor-file-name/U[:dev][/W][/Y]

where:

outdevice:*	specifies the volume to which you are copying the boot blocks. This specification must be the same as the indevice specification. The asterisk is required along with the device specification.
indevice	specifies the volume from which you are copying the boot blocks.
monitor-file-name	specifies the boot blocks for the monitor and handler you want to boot.
:dev	specifies the target system device name if it is different from the input device. For example, you can use this argument when you are creating a bootable RX01 diskette if the current system is on an RX02 system.
/W	initiates the copy operation and then pauses to let you change volumes.
/Y	ensures immediate execution of the copy operation by inhibiting confirmation messages.

NOTE

To be able to copy the boot-block information into the boot blocks of a volume, you need two files on that volume:

- The monitor file for the monitor you want to boot.
- The handler file for the device on which you want to boot.

  For example, for a double-density diskette system, check to see that the file DY.SYS is in the diskette directory. If it is, then you can copy the desired monitor onto the diskette, using the /U option.

Procedure for Making a Disk Bootable

The following procedure describes how to prepare a disk and make it bootable:

1. Obtain a formatted disk. (Most disks and diskettes are formatted by the manufacturer. However, Chapter 11 on the Format Utility (FORMAT) outlines the procedure for reformatting RK05, RK06, RK07, RP02, and RP03 disks, and RX01 and RX02 diskettes.)
2. Initialize the disk with /Z.
3. Copy the desired files onto the disk.
4. Copy the monitor and handler onto the disk.
5. Copy the monitor bootstrap into the boot blocks on the disk by using the /U option.

Example

The following example illustrates the preceding procedure and shows how to initialize a diskette, copy files to it, and write a bootstrap on the diskette:

1. The first command (step 2 of the procedure) initializes the diskette:

   ```
   *DU1:/Z/Y RET
   ```

2. The second command (which combines steps 3 and 4 of the procedure) squeezes all the files from DU0 onto DU1:

   ```
   *DU1:*=DU0:/S RET
   ```

3. The last command (step 5 of the procedure) writes the bootstrap for the FB monitor onto the bootstrap blocks (blocks 0 and 2–5) of DU1:

   ```
   *DU1:*=DU0:RT11FB.SYS/U RET
   ```

   The file name * is not significant; it is a dummy file name required by the Command String Interpreter.

Volume-ID Option (/V[:ONL])

The /V option displays the volume ID of a device and/or changes the volume ID.

Syntax

device:[/Z]/V[:ONL][/W][/Y]

where:

device:	is the device whose volume ID you want to display or change
/V	(if you specify only /V,) DUP displays on the terminal the volume ID and owner name of the device you specify.
/Z	(if you specify /Z with /V,) DUP initializes the device and prompts you for a new volume ID and owner name.
/Z/V:ONL	(if you specify /Z/V:ONL,) DUP assumes you want *only* to change the volume ID and owner name and not initialize the device. The /Z/V:ONL command changes only the volume ID and owner name; it does not initialize the device. See discussion on using /V with the /Z option to initialize a device and write new volume identification on it. You cannot change the volume ID of a magtape without initializing the entire tape.
/W	initiates the operation and then pauses to let you change volumes.
/Y	ensures immediate execution of the operation by inhibiting confirmation messages.

When you specify either /Z/V or /Z/V:ONL, DUP prompts you for a volume ID. For example:

```
Volume ID?
```

Respond with a volume ID that is up to 12 characters long for an RT–11 directory-structured volume or up to 6 characters long for a magtape. End your response by pressing RETURN. DUP then prompts for an owner name. For example:

```
Owner?
```

Respond with an owner name that is up to 12 characters long for an RT–11 directory-structured volume or up to 10 characters long for a magtape. End your response by pressing RETURN. DUP ignores characters you type beyond the valid length.

Example

This command writes a new volume ID and owner name on device DL1:

```
*DL1:/Z/V:ONL RET
DL0:/Volume ID change; Are you sure? Y RET
Volume ID? FORTRAN VOL RET
Owner?     Nancy RET
```

Wait-for-Volume Option (/W)

The /W option causes DUP to prompt you for the volumes to operate on, and waits for you to mount them. It is useful for single-disk systems or diskette systems. /W is a mode option that you can use with any of the action options, but you can specify only one action with it in a command line.

The /W option initiates execution of a command, but then pauses and displays the message *Mount input volume in <device>; Continue?*, where *<device>* represents the device into which you mount the input volume. At this time you can remove the system volume (if necessary) and mount the volume on which you actually want the operation to take place. When the new volume is loaded, enter Y or enter any string beginning with Y and then press RETURN to execute the operation.

If you enter N or any string beginning with N, or press CTRL/C, the operation is not completed. Instead DUP prompts you to remount the system volume, if you have removed it, and returns control to the keyboard monitor. Any other response causes the message to repeat.

If you enter Y, DUP prompts you for the input volume, if any. When the operation completes (except the /O operation, which boots the system), the *Mount system volume in <device>; Continue?* message is displayed. Replace the system device and enter Y or any string beginning with Y followed by RETURN. If you give any other response, DUP prompts you to mount the system volume until you enter Y. When you enter Y, the asterisk (*) prompt is displayed, and DUP waits for you to enter another command.

Example

The following command uses the /W option to scan a diskette for bad blocks:

```
*DU1:/K/F/W RET
Mount input volume in DU1; Continue? Y RET
?DUP-I-No bad blocks detected DU1:
Mount system volume in DU1; Continue? Y RET
*
```

During the first pause, the system diskette is removed and another diskette is mounted. The user then entered a Y RETURN to execute scan operation.

During the second pause, the user replaced the system disk (on which DUP is stored) and entered another Y RETURN to continue the operation.

Finally DUP prompts for another command.

When you use /W, make sure that DUP is on the system volume.

No-Query Option (/Y)

The /Y option suppresses the query messages that some commands display.

Some options (/C, /I, /O, /Q, /S, /T, and /Z) normally display the *Foreground job loaded, Continue?* message if a foreground job is loaded when you issue one of them. You must respond to the query message by typing Y or any string beginning with Y, and then press RETURN for the operation to proceed.

Some other options (/C, /I, /O, /S, /V, and /Z) display the *Are you sure?* message and wait for your response. If a foreground job is loaded and you specify one of these options, DUP combines the two query messages into one message and waits for your response.

You can suppress all these messages and the pause associated with them by specifying /Y in the command string.

If you use /Y with /Z to initialize your system volume, the system ignores /Y.

Directory-Initialization Option (/Z[:value])

The /Z option initializes a new directory or clears an old directory on an RT–11 formatted volume. This means the /Z option creates a directory structure on an RT–11 formatted volume or deletes file names from an old directory on an RT–11 formatted volume.

You must initialize a volume before you can store files on it. That is, the initialize operation must always be the first operation you perform on a new volume after you receive it, formatted, from a manufacturer. If the volume is not formatted, use the FORMAT utility to format the volume before you initialize it.

Initializing Magtapes

DUP initializes magtapes as well as disks and diskettes, though the initialization process is different for magtapes. When DUP initializes a magtape, it writes a label with a volume ID and owner name at the beginning of the tape.

Initialize magtapes for use with PIP (to copy files) and for use with programmed requests to FSM (MACRO–11 File Structured Modules for magtapes). Note, however, that it is not necessary to use DUP to initialize tape for use with BUP, since BUP initializes its own tapes.

RT–11 Directory Structure

RT–11 directories on disks and diskettes are divided into segments, each segment consisting of two blocks of disk space. RT–11 allows a maximum of 72 files for each directory segment, and 31 directory segments for each volume. So, the number of directory segments you have on a volume and the number of entries you fit in each segment determines the number of files you can list in a directory.

RT–11 does not normally support nonstandard directory structures, and Digital does not recommend altering the directory structure/format.

Syntax

device:/Z[:value][/N:value][/V][/R[:RET]][/B[:RET][/H]][/D][/W][/Y]

where:

device	specifies the volume you want to initialize.
/Z	initializes and/or clears a directory.
:value	is an octal integer (greater than or equal to 1) that specifies the size increase, in words, of each directory entry. DUP adds this number to the default number of words allocated for each entry (valid only for directory-structured volumes). If you do not specify a *value*, each entry is seven words long (for file name, creation date, and file position information).

When you allocate extra words, the number of entries for each directory segment decreases. The formula for determining the number of entries for each directory segment is:

```
(512-7)/((number of extra words)+7)
```

For example, if you use /Z:1, you can make 63 entries for each segment.

/N:value enables changing the default size of a directory when you initialize the volume.

/V lets you change the volume ID and owner name of a volume when you initialize the volume.

/R[:RET] (if you have RK06, RK07, RL01, or RL02 disks,) you can use the /R (replacement-table) option with /Z.

Use this option with /Z to scan a disk for bad blocks.

/B[:RET] covers bad blocks by scanning a volume for bad blocks and writing files over them when you initialize the volume.

/H used with /B to clear bad blocks caused by soft errors on MSCP class devices.

/D uninitializes (restores) a volume if you have not transferred any files to it since initialization.

/W starts the initialization operation and then pauses to let you mount the volume you want to initialize.

/Y suppresses confirmation messages to ensure immediate execution of the initialization.

CAUTION

Since /Z deletes directory entries from a directory, check a volume's directory for any files you want to save before you initialize it.

Usage

Creates an RT–11 directory structure (/Z)
Clears a directory of file names (/Z)
Increases the size allocated to each directory entry (/Z:value)
Changes the default directory size (/Z/N:value)
Changes volume ID and/or owner name (/Z/V)
Replaces bad blocks (/Z/R[:RET])
Covers bad blocks (/Z/B[:RET])
Restores a disk (/Z/D)

Example

The following command initializes the directory on the volume in device DU1:

```
*DU1:/Z RET
```

Changing Default Directory Size (/Z/N:value)

If you do not want the default directory size of a volume, use the /N option with /Z to set the desired number of directory segments for entries in the directory.

In this option, n is an octal integer in the range 1–31 that specifies the number of directory segments you want the directory to have.

Table 7–3 lists the default directory sizes, in segments, for directory-structured devices supported by RT–11.

Table 7–3: Default Directory Sizes

Device	Number (decimal) of Segments in a Directory
DL (RL01)	16
DL (RL02)	31
DM	31
DU (disk)	31
DU (diskette)	1
DW (RD50)	16
DW (RD51)	31
DX	1
DY (single-density)	1
DY (double-density)	4
DZ (RX50)	4
RK	16

If the default directory size for diskettes is too small for your needs, see the *RT–11 Installation Guide* for details on increasing the default number of directory segments.

The number of default segments in a directory depends on the size of the volume. Table 7–4 shows the algorithm RT–11 uses for determining directory segments.

Table 7–4: Algorithm for Determining Number of Directory Segments

Device Block Size = s					Default Number of Directory Segments
		s	$\leq$	512_{10}	1
512_{10}	<	s	$\leq$	2048_{10}	4
2048_{10}	<	s	$\leq$	12288_{10}	16_{10}
		s	>	12288_{10}	31_{10}

Example

The following command initializes the directory on device DL1 and allocates six directory segments to that directory:

```
*DL1:/Z/N:6 RET
DL1:/Initialize; Are you sure? Y RET
```

Changing Volume ID and/or Owner Name (/Z/V)

When you initialize a initialization time you want to change the volume ID and owner name, use the /V option with /Z. DUP then prompts you for the volume ID and owner name as a part of the initialization process.

Example

The following command initializes device DL1 and prompts you for a volume ID and owner name:

```
*DL1:/Z/V RET
DL1:/Initialize; Are you sure? Y RET
Volume ID? VOUCHERS RET
Owner? PAYABLES RET
```

See the Volume-ID Option (/V[:ONL]) section for examples of these prompts and instructions on how to use them.

Replacing Bad Blocks (/Z/R[:RET])

If you have RK06, RK07, RL01, or RL02 disks, you can use the /R (replacement-table) option with /Z.

Use this option with /Z to scan a disk for bad blocks. If DUP finds any bad blocks, it creates a replacement table so that routine operations access good blocks instead of bad ones. Thus, the disk appears to have only good blocks. Note, though, that accessing this replacement table slows response time for routine input and output operations.

DUP consults internal tables when processing commands, except in the case of the COPY/DEVICE command as follows:

- DUP no longer consults internal tables to determine if a device supports a software (DL and DM type) bad-block replacement table. The handlers that support such a replacement table should include the .DREST macro and specify the "replace" parameter.
- DUP does not consult internal tables to determine if a device returns an extra error word on absolute read/write special functions (377 and 376). Handlers for devices that return such an extra error word should include the .DREST macro and specify the DVM.DM argument for the mod parameter.
- DUP continues to consult internal tables to determine if a device is a magtape.

Previously, using /R:RET, DUP initialized the volume and retained the bad-block replacement table (and FILE.BAD files) created by the previous /R command. The /R:RET option is no longer supported with the /I option; COPY/DEVICE/RETAIN is not supported. The /R:RET option is ignored.

Note that the monitor file cannot reside on a block that contains a bad sector error (BSE) if you are doing bad-block replacement. If this condition occurs, a boot error results when you attempt to bootstrap the system. If this occurs, move the monitor.

With an RK06, RK07, RL01, or RL02 you have the option of deciding which bad blocks you want replaced if the number of bad blocks exceeds what can fit in the replacement table (replacement table overflow). The RK06s and RK07s support up to 32(decimal) bad blocks in the replacement table; the RL01s and RL02s support up to 10.

- With an RK06 or RK07 disk, DUP can replace only those bad blocks that generate a BSE. Of the blocks DUP cannot replace, DUP can report a bad block as being hard or soft. If you perform two bad block scans and a block is reported as bad in both reports, this indicates a hard error. If in the second report the block is not reported as bad, the block has recovered from a soft error.
- With an RL01 or RL02, DUP can replace any kind of bad block.

Example

The following paragraphs show how to designate which blocks to replace on an RK06, RK07, RL01, or RL02 disk.

When you use /R, DUP displays a list of replaceable bad blocks as in the following sample:

```
   Block            Type
030722  12754.   Replaceable
115046  39462.   Replaceable
133617  46991.   Replaceable
136175  48253.   Replaceable
136277  48319.   Replaceable
136401  48385.   Replaceable
140405  49413.   Replaceable
146252  52394.   Replaceable
?DUP-W-Bad blocks detected 8.
```

If there is a replacement table overflow, DUP prompts you to indicate which blocks you want replaced as follows:

```
?DUP-W-Replacement table overflow DEV:
Enter [RET], 0, or nnnnnn [RET]
Replace block #
```

The value nnnnnn represents the octal block number of the block you want the system to replace.

After you enter a block number, DUP responds by repeating the `Replace block #` prompt. Enter a 0 at any time if you do not want any more blocks replaced, and this will end prompting. DUP marks any blocks not placed in the replacement table as FILE.BAD.

If you press [RETURN] at any time, DUP places all bad blocks you have not entered into the replacement table, starting with the first on the disk, until the table is full. DUP assigns the name FILE.BAD to any remaining bad blocks and prompting ends.

If you use /Y with /R, the effect will be as if you pressed [RETURN] in response to the first `Replace block #` prompt.

Covering Bad Blocks (/Z/B[:RET])

To scan a volume for bad blocks and write files over them, use the /B option with /Z. For every bad block DUP encounters on the device, it creates a file called FILE.BAD to cover it. After the disk is initialized and the scan completed, the directory consists only of FILE.BAD entries that cover the bad blocks. If DUP finds a bad block in the boot block or the directory, it displays an error message and the disk is not usable.

You can use the /H option with /B to clear bad blocks caused by soft errors on MSCP class devices.

Retaining previously marked bad blocks:

- If you specify :RET with /B, DUP retains (when it initializes a volume) all FILE.BAD files created by a previous /B.
- If you use the /B option to initialize a volume that has been previously initialized using the /R option, DUP creates FILE.BAD files to cover the bad blocks on the volume, and the system then ignores the bad block replacement table.

  If the volume being initialized contains bad blocks, RT–11 displays the number of bad blocks and the locations of the bad blocks in octal and in decimal.

Example

The following command initializes a volume, discovers bad blocks on it, and displays their locations:

```
*DU0:/Z/B RET
DU0:/Initialize: Are you sure? Y RET
     Block         Type
000120     80.  Hard
000471    313.  Hard
000521    337.  Hard
?DUP-W-Bad blocks detected 3.
```

The left column lists the locations in octal, the middle column lists the locations in decimal, and the right column shows the type of bad block found: hard or soft.

Restoring a Disk (/Z/D)

Use the /D option with /Z to uninitialize (restore) a volume, if you have not transferred any files to it since initialization. DUP will restore all files and directory entries that were present before the volume was initialized. This option is useful if you initialize a volume by mistake. However, you cannot restore volumes that support bad-block replacement if bad blocks were found during initialization.

Because /D does not restore boot blocks, if you use /D to restore a previously bootable volume, you must use the bootstrap-copy option, /U[:dev], to make the volume bootable again.

Example

The following command restores volume DU1:

```
*DU1:/Z/D RET
```

DCL Equivalents of DUP Utility Options

Table 7–5 lists alphabetically DUP options and option combinations that have DCL commands using their functions:

- Column one contains DUP action options, all of which can be used by themselves with the exception of the /D option.
- Column two contains DUP options or syntax that can be combined with the option immediately preceding them in column one. Only those DUP options and option combinations having DCL equivalents are listed here.
- Column three contains the DCL commands that equal the DUP options in column one.
- Column four lists DCL command options that can be combined with the preceding DCL command in column three.

  These options are equivalent to the DUP options in column two.

Table 7–5: DCL Equivalents of DUP Utility Options

DUP Option	DUP Option Combination	DCL Command	DCL Command Option
/C		CREATE	
	[size]=		/ALLOCATE:size
	/G:value		/START:value
/I		COPY/DEVICE	
	/E:value		/END:value
	/F		/FILES
	/G:value		/START:value
	/H		/VERIFY
	/J		/IGNORE
	/W		/WAIT
	/Y		/NOQUERY
/K		DIRECTORY/BADBLOCKS	
	/E:value		/END:value
	/G:value		/START:value
	/F		/FILES
	/W		/WAIT
/O		BOOT	
	/Q		/FOREIGN

Table 7–5 (Cont.): DCL Equivalents of DUP Utility Options

DUP Option	DUP Option Combination	DCL Command	DCL Command Option
	/W		/WAIT
/S		SQUEEZE	
	device=		/OUTPUT:device
	/W		/WAIT
	/Y		/NOQUERY
/T:value		CREATE/EXTENSION:value	
/U[:dev]		COPY/BOOT[:dev]	
	/W		/WAIT
	/Y		/NOQUERY
/V		DIRECTORY/VOLUMEID:ONLY	
	/W		/WAIT
	/Y		/NOQUERY
/Z		INITIALIZE	
	/B[:RET]		/BADBLOCKS[:RET]
	/N:value		/SEGMENTS:value
	/R[:RET]		/REPLACE[:RET]
	/V[:ONL]		/VOLUMEID[:ONLY]
	/W		/WAIT
	/Y		/NOQUERY
/Z/D		INITIALIZE/RESTORE	
	/W		/WAIT
	/Y		/NOQUERY

NOTE

The default values for /ALLOCATE:size, END:value, EXTENSION:value, N:value, SEGMENTS:value, [size], START:value, and T:value are decimal while the default values for E:value, G:value, and Z:value are octal.

Chapter 8
Single-Line Text Editor (EDIT)

EDIT is a *single-line* text editor used for hardcopy terminals. See the *PDP–11 Keypad Editor User's Guide* for a complete description of the distributed keypad *screen* text editor. See the *Introduction to RT–11* for a tutorial on using KED/KEX.

Do not confuse the EDIT editor with the DCL EDIT command. See the description of the DCL EDIT command in the *RT–11 Commands Manual* for summary information on how to run all the editors that RT–11 supports.

Do not confuse EDIT with SL, the single-line *command* editor for issuing commands. See the *Introduction to RT–11* for a description of SL.

The EDIT editor allows you to edit only one line of text at a time. For this reason, EDIT is useful for editing files on a hardcopy terminal. However, EDIT reads ASCII text and files from any input device, and writes on any output device.

Calling Edit

The default RT–11 text editor is the KED/KEX screen editor. You can call the single-line text editor EDIT in either of the following two ways:

- Assign the DCL EDIT command to run EDIT.SAV (the single-line text editor) by issuing the command:

  ```
  .SET EDIT EDIT
  ```

  Then issue the DCL EDIT command. The syntax for issuing the DCL EDIT command is:

  ```
  EDIT[/option filespec[/option]
  ```

- Issue the DCL EDIT command with the EDIT editor option:

  ```
  .EDIT/EDIT filespec[/option]
  ```

Running EDIT

When you want to edit an existing file, the EDIT editor does not perform any I/O operations as a result of your DCL EDIT command. You must issue the R command to the editor, after you call it, to read the first page of text and make it available for you to work on. The following example invokes EDIT, opens an existing file, and reads the first page of text:

```
.EDIT MYFILE.TXT RET
*R$$
```

It is possible to receive an error or warning message as a result of the EDIT command. If, for example, the file you need to edit is not on device DK, EDIT issues an error message, but remains in control. When a situation like this occurs, you can either issue another command directly to EDIT or type CTRL/C ESC ESC to return control to the monitor; for example:

```
.EDIT/INSPECT EXAMP3.TXT RET
?EDIT-F-file not found
* CTRL/C ESC ESC
.
```

EDIT Reads a File in Units Called Pages

EDIT considers a file to be divided into logical units called pages. A page of text is generally 50 to 60 lines long (delimited by form-feed characters) and corresponds approximately to a physical page of a program listing. The editor reads one page of text at a time from the input file into its internal text buffers, where the page becomes available for editing.

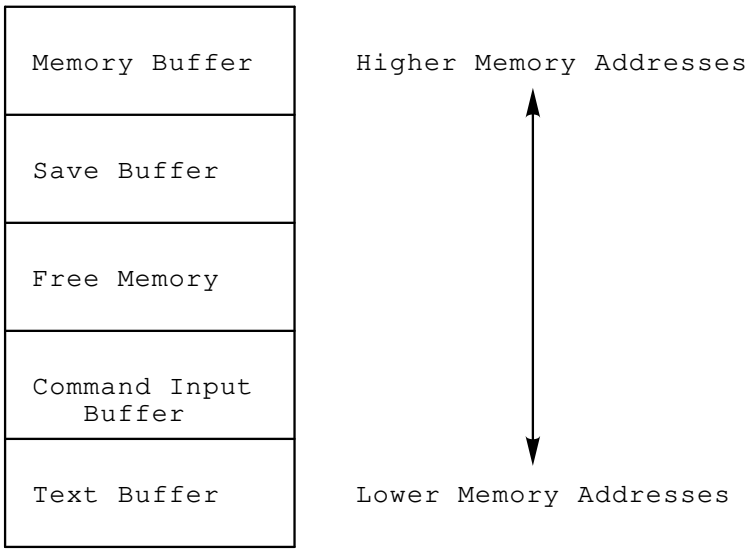
Location Pointer

Most EDIT commands function with respect to a movable location pointer that is normally located between the most recent character operated on and the next character in the buffer. It is important to think of this pointer as being between two characters, and never directly on a character.

At the start of editing operations, the pointer precedes the first character in the buffer, although it is not displayed on the terminal. At any time during the editing procedure, think of the pointer as representing the current position of the editor in the text. The pointer moves during editing operations according to the type of editing operation being performed. Refer to text in the buffer as so many characters or lines preceding or following the pointer.

Memory Usage

The memory area used by the editor is divided into four logical buffers as follows:



The text buffer contains the current page of text you are editing, and the command input buffer holds the command you are currently typing at the terminal. If a command you are currently entering is within ten characters of exceeding the space available in the command buffer, the following message displays on the terminal:

```
?EDIT-W-Command buffer almost full
```

If you can complete the command within ten characters, you can finish entering the command; otherwise, you should press ESCAPE twice to execute that portion of the command line already completed. The warning message displays each time you enter a character in one of the last ten spaces.

If you attempt to enter more than ten characters, EDIT displays the following message and aborts the entire command:

```
?EDIT-F-Command buffer full; no command(s) executed
```

The save buffer contains text stored with the Save (S) command, and the macro buffer contains the command string macro entered with the Macro (M) command. (Both commands are explained in the Utility Commands section.)

EDIT does not allocate space for the macro and save buffers until an M or S command executes. Once you enter an M or S command, a subsequent 0M or 0U (Unsave) command executes to return that space to the free area.

The size of each buffer automatically expands and contracts to accommodate the text you are entering; if there is not enough space available to accommodate required expansion of any of the buffers, EDIT displays the error message:

```
?EDIT-F-Insufficient memory
```

Two Modes of Operating EDIT: Command and Text

The editor operates in either command mode or text mode.

Command Mode

In command mode, EDIT interprets all input you type on the keyboard as commands to perform some operation.

Immediately after being loaded into memory and started, the editor is in command mode. EDIT displays an asterisk at the left margin of the terminal page to indicate that it is ready to accept a command.

You execute commands by pressing ESCAPE twice in succession.

Execution of commands proceeds from left to right. When EDIT encounters an error before beginning execution of a command string, it displays an error message followed by an asterisk at the beginning of a new line, indicating that it is still in command mode, that it is waiting for a command, and that execution of the illegal command has not occurred. You should retype the command correctly.

Text Mode

In text mode, EDIT interprets all typed input as text to insert into, replace with, or append to the contents of the text buffer.

To enter text mode, you must issue a command followed by a text string. When you issue one of these commands, EDIT recognizes all succeeding characters as part of the text string until it encounters an ESCAPE character. Pressing ESCAPE once terminates the text string and causes EDIT to reenter command mode.

Special EDIT Key Commands

The following table lists special keys that EDIT uses as commands.

Key	Function
ESCAPE, ALTMODE, or SEL	ESC key; echoes as $. Pressing ESCAPE once terminates a text string. Pressing ESCAPE twice executes a command or command string. For example: `*GMOV A,B$-1D$$` The first ESCAPE ($) terminates the text object (MOV A,B) of the Get command. The double ESCAPE terminates the Delete command and causes execution of the entire statement with the result that the character B will be deleted.

CTRL/C	Echoes as ^C. When in EDIT command mode, press CTRL/C once to terminate EDIT and return control to the monitor. To restart EDIT, type R EDIT or REENTER in response to the monitor's prompt.
	When in EDIT text mode, a CTRL/C is included in the text, just like any other character.
	If the editor is executing a lengthy command and you want to stop EDIT, type CTRL/C twice. This aborts the command, generates the *?EDIT-F-Command aborted* error message, and returns EDIT to command mode. For example:
	`*I^C^C^C$$`
	`*^C$$`
	In the first command, the three CTRL/C characters are part of the text object of the Insert command. EDIT treats them like any other character. In the second command string, the CTRL/C occurs at command level, which causes the editor to terminate.
	If no commands (other than CLOSE) are executed between the time you terminate the editor and the time you issue a REENTER command, the text buffer is preserved as it was at program termination. However, only the text buffer is preserved. The input and output files are closed, and the save and macro buffers are reinitialized.
	If you inadvertently terminate an editing session before the output file can be closed, you can often use the monitor CLOSE command to make permanent the portion of the output file that has already been written (see the *RT–11 Commands Manual*). You can then reenter the editor, open a new output file, and continue the editing session.

CTRL/O	Echoes as ^O and a return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL/O resumes output.

Special EDIT Key Commands

Key	Function
CTRL/U	Echoes as ^U and a return. Deletes all characters on the current terminal input line. (Typing CTRL/U has the same effect as pressing <X until all the characters back to the beginning of the line are deleted.)
<X DELETE RUBOUT	Deletes a character from the current command line; echoes as a backslash (\) followed by the character deleted. Each succeeding time you press <X a character is deleted and the terminal echoes that character. An enclosing backslash displays when you type a key other than <X. This erasure is done from right to left. Since EDIT accepts multiple-line commands, <X can delete past the return and line-feed combination and delete characters on the previous line. You can use <X in both command and text modes.
TAB	Spaces to the next hardware tab stop. Tab stops are positioned every eight spaces on the terminal; pressing TAB causes the cursor or carriage to advance to the next tab position.

CTRL/X	Echoes as ^X and a return. Pressing CTRL/X causes EDIT to ignore the entire command string you are currently entering. EDIT displays a return and line-feed combination and an asterisk to indicate that you can enter another command. For example:
	
	`*IABCD`
	`EFGH^X`
	`*`
	
	If, in the preceding example, you pressed CTRL/U, you would have deleted only EFGH; but pressing CTRL/X deletes the entire command.
	
	If you are running a system job, you must issue the SET TT:NOFB command to enable the CTRL/X function. If you do not and you press CTRL/X, RT–11 intercepts the CTRL/X and prompts you for a system-job name.

Command-Line Syntax

The general syntax for all EDIT commands is:

[argument]command[text] ESC

or

[argument]command ESC

where:

argument is one of the following arguments.

Argument	Meaning
n	specifies an integer in the range -16383 to +16383 (decimal) and may, except where noted, be preceded by a plus (+) or minus (-) sign. If no sign precedes *n*, it is assumed to be a positive number. The absence of *n* implies a 1 (or -1 if a minus sign precedes a command). This argument can represent the number of characters or lines forward (+) or backward (-) to move the pointer, or it can represent the number of times to execute the operation.
0	specifies the text between the beginning of the current line and the location pointer.
/	specifies the text between the location pointer and the end of the text in the buffer.
=	specifies -*n*, where *n* is equal to the length of the last text argument used. Use this argument with the Jump, Delete, and Change commands only.

command specifies a 1- or 2-letter command.

text specifies a string of ASCII characters.

Character- and Line-Oriented Commands

EDIT commands are either character-oriented or line-oriented.

Character-Oriented Commands

Character-oriented commands affect a specified number of characters preceding or following the pointer.

Command Argument

The argument to character-oriented commands specifies the number of characters in the buffer on which to operate. If the argument is unsigned (positive), the command moves the pointer in a forward direction. If *n* is preceded by a minus sign (negative), the command moves the reference pointer backward.

Line Feeds, Returns, and Null Characters

Line feeds, returns, and null characters, although not printed, are embedded in text lines, counted as characters in character-oriented commands, and treated as any other text characters.

When you press RETURN, both a return and a line-feed character are inserted into the text. For example, assume the pointer is positioned as indicated in the following text (↑ represents the current position of the pointer):

```
MOV   VECT,R2 RET LF ↑
CLR   @R2 RET LF
```

The EDIT command -2J moves the pointer back two characters to precede the return character:

```
MOV   VECT,R2↑  RET LF
CLR   @R2  RET↑  LF
```

The command 10J advances the pointer forward ten characters and places it between the RET and LF characters at the end of the second line. Note that the tab character preceding @R2 is also counted as a single character:

```
MOV   VECT,R2 RET LF
CLR   @R2 RET↑  LF
```

Finally, to place the pointer after the C in the first line, use a -14J command (see the description of the J (Jump) command in the Pointer-Relocation Commands section):

```
MOV   VECT↑ ,R2 RET LF
CLR   @R2 RET LF
```

Line-Oriented Commands

Line-oriented commands operate on entire lines of text.

Command Argument

When you use line-oriented commands, the command argument specifies the number of lines on which to operate. Because EDIT counts the line-terminating characters to determine the number of lines on which to operate, the command argument does not affect the same number of lines forward (positive) as backward (negative).

For example, the argument -1 applies to the line beginning with the first character following the second previous end-of-line and ending with the character preceding the pointer. The argument 1 in a line-oriented command, however, applies to the text beginning with the first character following the pointer and ending at the first end-of-line. Thus, if the pointer is at the center of the line, the argument -1 affects one and one-half lines backward from the pointer and the argument 1 affects one-half line beyond the pointer.

For example, assume the buffer contains:

```
MOV     ↑ PC,R1 RET LF
ADD     #DRIV-.,R1 RET LF
MOV     #VECT,R2 RET LF
CLR     @R2 RET LF
```

The command to advance the pointer one line (1A) causes the following change:

```
MOV     PC,R1 RET LF
↑ ADD   #DRIV-.,R1 RET LF
MOV     #VECT,R2 RET LF
CLR     @R2 RET LF
```

The command 2A moves the pointer over two RET LF combinations to precede the fourth line:

```
MOV     PC,R1 RET LF
ADD     DRIV-.,R1 RET LF
MOV     VECT,R2 RET LF
↑ CLR   @R2 RET LF
```

For another example, assume the buffer contains:

```
MOV     PC,R1  RET LF
ADD     DRIV-.,R1  RET LF
MOV     VEC↑ T,R2  RET LF
CLR     @R2 RET LF
```

A command of -1A moves the pointer back by one and one-half lines to precede the second line:

```
MOV     PC,R1 RET LF
↑ ADD  DRIV-.,R1 RET LF
MOV     VECT,R2 RET LF
CLR     @R2 RET LF
```

A command of -1A moves the pointer back by only one line:

```
↑ MOV   PC,R1 RET LF
ADD     DRIV-.,R1< RET LF
MOV     VECT,R2 RET LF
CLR     @R2 RET LF
```

Repeating Commands or Command Strings

You can execute commands, command strings, or portions of a command string more than once by enclosing the portions or command in angle brackets (<>) and preceding the left angle bracket with the number of times you want the command to repeat. The syntax is: **n<command>**

A syntax example of a command string is:

```
c1$c2$n<c3_$c4_$>c5$$
```

where:

c	specifies a command. The number after the command indicates its order: first, second, and so on.
n	specifies the number of times to repeat the command string surrounded by < and >.

In the preceding syntax example, commands C1 and C2 each execute once, then commands C3 and C4 execute *n* times. Finally, command C5 executes once and the command line is finished.

The iteration argument (*n*) must be a positive number (in the range 1 through 16383 decimal); if unspecified, it is assumed to be 1. If the number is negative or too large, an error message displays. You can nest iteration brackets up to 20 levels. Before execution, EDIT checks command lines to make certain the brackets are correctly used and match.

Enclosing a portion of a command string in iteration brackets and preceding it with an iteration argument (*n*) has the same result as typing that portion of the string *n* times. Thus, these two examples are equivalent:

```
*BGAAA$3<-DIB_$-J>V$$
```

```
*BGAAA$-DIB$-J-DIB$-J-DIB$-JV$$
```

Similarly, the following two strings are equivalent:

```
*B3<2<AD>V>$$
```

```
*BADADVADADVADADV$$
```

The following bracket structures are examples of legal usage:

```
<<><<<><>>>>
<<<>>><><>
```

The next bracket structures are examples of combinations that cause an error message:

```
><><
<<<>>
```

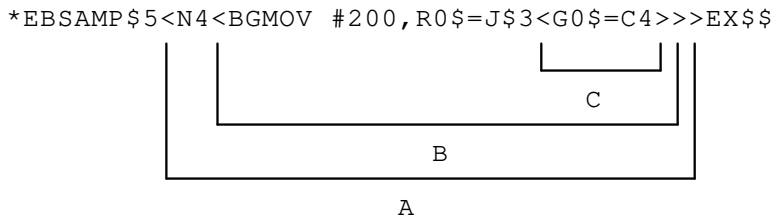
During command repetition, execution proceeds from left to right until a right bracket is encountered. EDIT then returns to the last left bracket encountered, decreases the iteration counter, and executes the commands within the brackets. When the counter is decreased to 0, EDIT looks for the next iteration count to the

left and repeats the same procedures. The overall effect is that EDIT works its way to the innermost brackets and then works its way back again.

The most common use for iteration brackets is found in commands, such as Unsave (U), that do not accept repeat counts. For example:

```
*3>U>$$
```

Assume you want to read a file called SAMP (stored on device DK), and you want to change the first four occurrences of the instruction MOV 200,R0 on each of the first five pages to MOV 244,R4. Enter the following command line:



This command line contains three sets of iteration loops (A, B, C) and executes as follows:

1. Execution initially proceeds from left to right; EDIT opens the file SAMP for input and reads the first page into memory.
2. EDIT moves the pointer to the beginning of the buffer and initiates a search for the character string MOV 200,R0.
3. When it finds the string, EDIT positions the pointer at the end of the string, but the =J command moves the pointer back, so that it is positioned immediately preceding the string.
4. At this point, execution has passed through each of the first two sets of iteration loops (A, B) once. The innermost loop (C) is next executed three times, changing the 0s to 4s.
5. Control then moves back to pick up the second iteration of loop B, and again moves from left to right.
6. When loop C has executed three times, control again moves back to loop B.
7. When loop B has executed a total of four times, control moves back to the second iteration of loop A, and so forth, until all iterations have been satisfied.

Summary of Rules for Entering Commands

- To execute a command or a command string, press ESCAPE twice.
- To terminate a text string, press ESCAPE once.

  You can separate commands from one another by a single ESCAPE; however, if the command requires no text, the separating ESCAPE is not necessary.

- Type an argument to a command before the command letter.

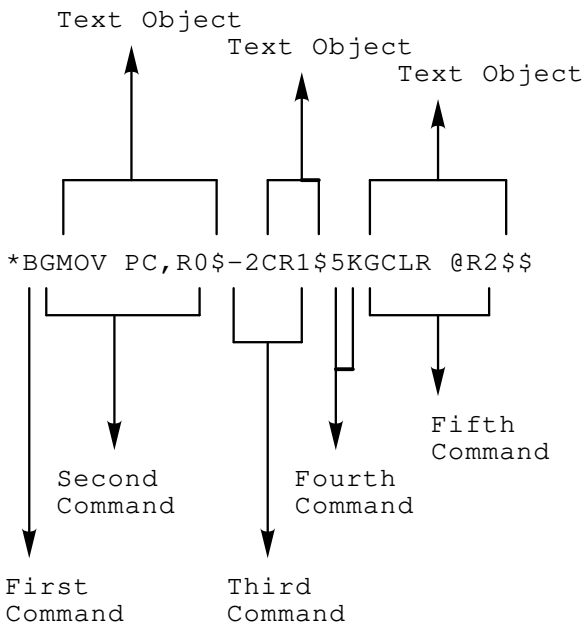
  Arguments specify one of two things:

  — The number of times to perform the command.

  — The particular portion of text to be affected by the command.

  With some commands, this specification is implicit and no argument is needed; other editing commands require an argument. See the descriptions of the editing commands for the descriptions of their individual arguments.

- You can use spaces, returns, and line feeds within a command string to increase command readability. EDIT ignores them unless they appear in a text string.
- You can place text strings that are several lines long within commands to insert text. Press RETURN to terminate each line you enter. That inserts both a return and a line-feed character into the text. Execute the entire command by pressing ESCAPE twice.
- You can string several commands together and execute them in sequence. Consider the following example:



The $ in the example indicates the ESCAPE character (pressing ESCAPE once). This character separates the end of each text object from the next command. The $$ specifies two ESCAPE characters (pressing ESCAPE twice). Execution of a command string begins when you type the double ESCAPE and proceeds from left to right.

- EDIT ignores spaces, returns, line feeds, and single ESCAPEs, except when they are part of a text string. Thus, these two examples produce the same result:

```
*BGMOV R0$=CCLR R1$AV$$
```

```
*B$ GMOV R0$
=CCLR R1$
A$ V$$
```

- You can execute commands, command strings, or portions of a command string more than once by enclosing the portions or command in angle brackets (<>) and preceding the left angle bracket with the number of times you want the command to repeat.

EDIT Command Types and Descriptions

EDIT commands fall into seven general types. The following table lists these types and the commands they include. The following sections describe these commands according to the types contained in the table.

Type	Commands
File Open and Close	Edit Backup (EB) Edit Read (ER) Edit Write (EW) End File (EF)
File Input and Output	Exit (EX) Next (nN) Read (R) Write (nW)
Pointer-Relocation	Advance (nA) Beginning (B) Jump (nJ)
Search	Find (nF) Get (nG) Position (nP)
Text-Listing	List (nL) Verify (V)
Text-Modification	Change (nC) Delete (nD) Exchange (nX) Insert (I) Kill (nK)
Utility	Edit Lower (EL) Edit Upper (EU) Edit Version (EV) Execute Macro (nEM) Macro (M)

File Open and Close Commands

You can use file open and close commands to:

- Open an existing file for input and prepare it for editing (ER).
- Open a file for output of newly created or edited text (EW).
- Open an existing file for editing and create a backup version of it (EB).
- Close an open output file (EF).

Edit Read (ER)

The Edit Read (ER) command opens an existing file for input and prepares it for editing. Only one file can be open for input at a time.

The syntax of the command is: **ERdev:filnam.typ**

The argument dev:filnam.typ is limited to 14_{10} characters and specifies the file to be opened. If you do not specify a device, DK: is assumed. If a file is currently open for input, EDIT closes that file and opens the new one.

Edit Read does not input a page of text nor does it affect the contents of the other user buffers.

With Edit Read you can close a file that is already open for input and reposition EDIT at the beginning of the file. The first Read command following any Edit Read command inputs the first page of the file.

Example

This command string opens the file SAMP.MAC on device DU1:

```
*ERDU1:SAMP.MAC$$
```

NOTE

If you enter EDIT with the monitor EDIT /INSPECT or EDIT/OUTPUT command, an Edit Read command is automatically performed on the file named in the EDIT command.

Edit Write (EW)

The Edit Write (EW) command opens a file for output of newly created or edited text. However, no text is output and the contents of the buffers are not affected. Only one file can be open for output at a time. EDIT closes any output files currently open and preserves any edits made to the file.

The syntax of the command is: **EWdev:filnam.typ[n]**

The argument dev:filnam.typ[n] is limited to 21_{10} characters and is the name you assign to the output file being opened. If you do not specify a device, DK is assumed. The optional argument [n] is a decimal number that represents the length of the file to be opened. Note that the square brackets ([ ]) are part of this

argument and must be typed. If you do not specify [n], the system will default to either the larger of one-half the largest available space, or the second largest available space. If this is not adequate for the output file size, you must close this file and open another when this one becomes full. You should use the [n] construction whenever there is doubt as to whether enough space is available on the device for one output file.

If a file with the same name already exists on the device, EDIT deletes the existing file when you type an Exit, End File, or another Edit Write command. EDIT then displays the warning message:

```
?EDIT-W-Superseding existing file
```

Example

The following command, for example, opens FILE.BAS on device DK and allocates 11 blocks of space for it:

```
*EWFILE.BAS[11]$$
```

NOTE

If you enter EDIT with the monitor EDIT/CREATE command, an Edit Write command is automatically performed on the file named in the EDIT command. If you enter EDIT with the monitor EDIT/OUTPUT command, an Edit Write is automatically performed on the file named with the /OUTPUT option.

Edit Backup (EB)

The Edit Backup (EB) command opens an existing file for editing and at the same time creates a backup version of the file. EDIT closes any input and output files currently open. No text is read or written with this command.

The syntax of the command is: **EBdev:filnam.typ[n]**

The argument dev:filnam.typ[n] is limited to 21_{10} characters. If you do not specify a device, DK is assumed. The argument [n] is optional and represents the length of the file to be opened; if you do not specify [n], the system defaults to the larger of either one-half the largest available space or the second largest available space.

The file you indicate in the command line must already exist on the device you designate, because text will be read from this file as input. At the same time, EDIT opens an output file under the same file name and file type. When the output file is closed, EDIT renames the original file (used as input) with the current file name and a BAK file type, and deletes any previous file with this file name and a BAK file type. EDIT closes the new output file and assigns it the name you specify in the EB command. This renaming of files takes place when an Exit, End File, or subsequent Edit Write or Edit Backup command executes. If you terminate the editing session with a CTRL/C command before the output file is closed, the new output file is not made permanent, and the renaming of the current version to BAK does not take place.

Example

When editing is complete, the old BAS1.MAC becomes BAS1.BAK and the new file becomes BAS1.MAC. EDIT deletes any previous version of BAS1.BAK. This command opens BAS1.MAC on device SY:

```
*EBSY:BAS1.MAC$$
```

NOTE

In EB, ER, and EW commands, leading spaces between the command and the file name are not permitted because EDIT assumes the file name to be a text string. All dev:filnam.typ specifications for EB, ER, and EW commands conform to RT–11 conventions for file naming. File names entered in command strings used with other system programs have identical specifications.

If you enter EDIT with the monitor EDIT command, an Edit Backup command is automatically performed on the file named in the EDIT command.

End File (EF)

The End File (EF) command closes the current output file and makes it permanent. You can use the EF command to create an output file from a section of a large input file, or to close an output file that is full before you open another file. Modifiers are illegal with an EF command. Note that an implied EF command is included in the EW and EB commands.

The syntax of the command is: **EF**

File Open and Close Commands

The Relationship Between Open and Close Commands

The following table shows the relationship between the file open and close commands and the buffers and files.

Command	File Status	Input Text Buffer	Output
ERXXX	Opens XXX for input; closes existing input file, if any	Unchanged	Unchanged
EWXXX	Unchanged	Unchanged	Opens XXX for output; closes existing output file, if any; performs BAK renaming if EB is in effect
EBXXX	Opens XXX for input; closes existing input file, if any	Unchanged	Opens temporary file for output; closes existing output file, if any; performs BAK renaming if EB is in effect
EF	Unchanged	Unchanged	Closes output file; performs BAK renaming if EB is in effect
EX	Copies to output file	Copies to output file	Closes output file after copying complete; performs BAK renaming if EB is in effect

File Input and Output Commands

You use file input and output commands to:

- Read text from an input file into the buffer (R).
- Copy lines of text from the buffer into an output file (nW) (nN).
- Terminate the editing session (EX).

Read (R)

Before you can edit text, you must read the input file into the buffer. The Read (R) command reads a page of text from the input file (previously specified in an ER or EB command) and appends it to the current contents of the text buffer (contents can be empty).

The syntax of the command is: **R**

No arguments are used with the R command. If the buffer contains text when the R command is executed, the pointer does not move; however, if the buffer does not contain text, the pointer is placed at the beginning of the buffer.

The Condition for Ending a Transfer

EDIT transfers text to the buffer until one of the following conditions occurs:

- A form-feed character, signifying the end of the page, is encountered.
- The text buffer is 500 characters from being full. (When this condition occurs, the Read command inputs up to the next return and line-feed combination, then returns to command mode. An asterisk displays as though the read were complete, but text will not have been fully input.)
- An end-of-file is encountered. (The *?EDIT-F-End of input file* message displays when all text in the file has been read into memory and no more input is available.)

The maximum number of characters you can bring into memory with an R command depends on the system configuration and the memory requirements of other system components. EDIT displays an error message if the read exceeds the memory available or if no input is available.

Examples

1. This command opens SJK1.BAS on DK and permits modification:

```
*EBSJK1.BAS$$
```

2. This command reads the first page of SJK1.BAS into the buffer. The pointer is placed at the beginning of the buffer. The /L command lists the contents of the buffer on the terminal, beginning at the pointer and ending with the last character in the buffer:

```
*R/L$$
THISISPAGEONEOF
FILE SJK1.BAS.
```

Write (nW)

The Write (nW) command copies lines of text from the text buffer to the output file (as specified in the EW or EB command). The contents of the buffer are not altered and the pointer is left unchanged (unless an output error occurs).

NOTE

EDIT uses a system of intermediate buffers to store output before it writes the data to an output file. The Write command logically writes to the file, but output to a device does not occur until the intermediate buffer fills. When the editor closes a file (that is, after you issue an EF, EB, EX, or EW command), text is written from the buffer to the file and the file is complete. If the editor does not close a file (if you exit by typing CTRL/C and use the CLOSE command), it is possible that the output file will be missing the last 512 characters.

The syntax of the command is: **nW**

where:

n	determines the lines of text to copy, writing n lines of text, beginning at the pointer and ending with the nth end-of-line character, to the output file.
-n	writes n lines of text to the output file beginning with the first character on the -nth line and terminating at the pointer.
0	writes to the output file the current line up to the pointer.
/	writes to the output file the text between the pointer and the end of the buffer.

If the buffer is empty when the write executes, no characters are output.

Examples

1. This command writes the five lines of text following the pointer into the current output file:

```
*5W$$
```

2. This command writes the two lines of text preceding the pointer into the current output file:

```
*-2W$$
```

3. This command writes the entire text buffer to the current output file:

```
*B/W$$
```

NOTE

If an output file fills while a Write command is executing, EDIT displays the *?EDIT-F-Output file full*? message. In this case, EDIT positions the reference pointer after the last character it wrote successfully. You can then use the following recovery procedure:

1. Close the current output file (EF command).
2. Open a new output file (EW command).
3. Delete the characters just written by using -nD or -nK, where n is any arbitrary number that exceeds the number of lines or characters in the buffer.
4. Resume output.

Next (nN)

The Next (nN) command writes the contents of the text buffer to the output file, deletes the text from the buffer, and reads the next page of the input file into the buffer. The pointer is positioned at the beginning of the buffer.

The syntax of the command is: **nN**

If you specify the argument n with the Next command, the sequence is executed n times. The N command operates in a forward direction only; therefore, you cannot specify negative arguments with an N command.

If EDIT encounters the end of the input file when trying to execute an N command, it displays *?EDIT-F-End of input file* to indicate that no further text remains in the input file. Since the contents of the buffer have already been transferred to the output file, the buffer is empty.

Using the N command is a quick way to write edited text to the output file and set up the next page of text in the buffer. The N command functions as though it were a combination of the Write, Delete, Read, and Beginning commands. (Delete is a text modification command, described in the Text-Modification Commands

section. The Beginning command is a pointer relocation command, described in the Text-Modification Commands section. The N command with an argument is a convenient way to set up text in the buffer, if you already know its page location.)

Examples

1. In the following series of code examples, an N command copies an input file with more than one page of text to the output file. the first command opens the file TEST.MAC on device DK and creates a new file entitled TEST.MAC for output:

```
*EBDK:TEST.MAC$$
```

2. This command reads the first page of the input file, TEST.MAC, into the buffer and lists the entire page on the terminal:

```
*N/L$$
THIS IS PAGE ONE OF
FILE TEST.MAC
```

3. This command transfers the contents of the buffer to the output file, clears the buffer, and encounters the end of the file. Because it cannot complete the N sequence, EDIT displays *?EDIT-F-End of input file* on the terminal. The buffer is empty and the entire input file has been written to the output file:

```
*N/L$$
?EDIT-F-End of input file
*
```

Exit (EX)

Type the Exit (EX) command to terminate an editing session. The Exit command:

- Writes the text buffer to the output file.
- Transfers the remainder of the input file to the output file.
- Closes all open files.
- Renames the backup file with a BAK file type if an EB command is in effect.
- Returns control to the monitor.

The syntax of the command is: **EX**

No arguments are accepted. Essentially, Exit copies the remainder of the input file into the output file and returns to the monitor. Exit is legal only when there is an output file open. If an output file is not open and you want to terminate the editing session, return to the monitor by typing CTRL/C.

NOTE

You must issue an EF or EX command in order to make an output file permanent. If you press CTRL/C twice to return to the monitor without issuing an EF command, the current output file will not be saved.

(You can, however, make permanent that portion of the text file that has already been written out, by using the monitor CLOSE command.)

Example

The following example shows the contrasting uses of the EF and EX commands. Assume an input file, SAMPLE, contains several pages of text. In this example code, the first and second pages of the file are made into separate files called SAM1 and SAM2, respectively; the remaining pages of text make up the file SAMPLE:

```
*EWSAM1$$
*ERSAMPLE$$
*RNEF$$
*EWSAM2$$
*NEF$$
*EWSAMPLE$EX$$
```

Note that the EF commands are not necessary in this example, since the EW command closes a currently open output file before opening another.

Pointer-Relocation Commands

Pointer-relocation commands allow you to change the current location of the pointer within the text buffer to:

- The beginning of the buffer (B)
- A specified number of characters forward (nJ)
- A specified number of lines forward (nA)

Beginning (B)

The Beginning (B) command moves the current location of the pointer to the beginning of the text buffer.

This command has no arguments. The command's syntax is: **B**

Example

Assume the buffer contains:

```
MOVB    5(R1),@R2
ADD     R1,(R2)+
CLR     @↑ R2
MOVB    6(R1),@R2
```

The B command moves the pointer to the beginning of the text buffer:

```
*B$$
```

The B command makes the text buffer look like this:

```
↑ MOVB  5(R1),@R2
ADD     R1,(R2)+
CLR     @R2
MOVB    6(R1),@R2
```

Jump (nJ)

The Jump (nJ) command moves the pointer past a specified number of characters in the text buffer.

The syntax of the command is: **nJ**

where:

(+ or -)**n**	moves the pointer (forward or backward) n characters.
0	moves the pointer to the beginning of the current line (equivalent to 0A).
/	moves the pointer to the end of the text buffer (equivalent to /A).
=	moves the pointer backward n characters, where n equals the length of the last text argument used.

Negative arguments move the pointer toward the beginning of the buffer; positive arguments move it toward the end. Jump treats returns, line feeds, and form-feed characters the same as any other characters, counting one buffer position for each one.

Examples

1. This command moves the pointer ahead three characters:

   ```
   *3J$$
   ```

2. This command moves the pointer back four characters:

   ```
   *-4J$$
   ```

3. This command moves the pointer so that it immediately precedes the first occurrence of ABC in the buffer:

   ```
   *B$GABC$=J$$
   ```

Advance (nA)

The Advance (nA) command is similar to the Jump command, except that it moves the pointer a specific number of lines (rather than single characters) and leaves it positioned at the beginning of the line.

The syntax of the command is: **nA**

where:

n	moves the pointer forward n lines and positions it at the beginning of the nth line.
-n	moves the pointer backward past n return and line-feed combinations and positions it at the beginning of the -nth line.
0	moves the pointer to the beginning of the current line (equivalent to 0J).
/	Moves the pointer to the end of the text buffer (equivalent to /J).

Examples

1. This command moves the pointer ahead three lines:

   ```
   *3A$$
   ```

2. Assume the buffer contains:

   ```
   CLR @R2
   ```

   This command moves the pointer to the beginning of the current line:

   ```
   *0A$$
   ```

   Now the buffer looks like this: `CLR @R2`

Search Commands

Use search commands to locate characters or strings of characters within text. Search commands locate:

- The nth occurrence of a specified text string in the buffer (nG)
- The nth occurrence of a specified text string in the input file while transferring the buffer contents to the output file as each page is unsuccessfully searched (nF)
- The nth occurrence of a specified text string in the input file while deleting the buffer contents as each page is unsuccessfully searched (nP)

NOTE

Search commands always have positive arguments. They search ahead in the file. This means that to search for a character string that has already been written to the output file, you must first close the currently open files (with EX) and then edit the file that was just used for output (with EB).

Get (nG)

The Get (nG) command is the basic search command in EDIT. It searches the current text buffer for the nth occurrence of a specific text string, starting at the current location of the pointer. If you do not supply the argument n, EDIT searches for the first occurrence of the text object.

The search terminates when EDIT either finds the nth occurrence or encounters the end of the buffer. If the search is successful, EDIT positions the pointer to follow the last character of the text object. EDIT notifies you of an unsuccessful search by printing *?EDIT-F-Search failed*. In this instance, EDIT positions the pointer after the last character in the buffer.

The syntax of the command is: **nGtext**

The argument n must be positive. If you omit it, EDIT assumes it to be 1.

The text string may be any length and must immediately follow the G command. EDIT makes the search on the portion of the text between the pointer and the end of the buffer.

Examples

1. Assume the pointer is at the beginning of the buffer in this example:

```
↑ MOV   PC,R1
ADD     DRIV-.,R1
MOV     VECT,R2
CLR     @R2
MOVB    5(R1),@R2
ADD     R1,(R2)+
CLR     @R2
MOVB    6(R1),@R2
```

   The following command searches for the first occurrence of the characters ADD following the pointer and places the pointer after the searched characters:

```
*GADD$$
```

   After the preceding command is executed, the buffer looks like this:

```
MOV     PC,R1
ADD↑    DRIV-.,R1
```

2. This command searches for the third occurrence of the characters @R2 following the pointer and leaves the pointer immediately following the text object:

```
*3G@R2$$
```

   After the preceding command is executed, the buffer is changed to:

```
ADD   R1,(R2)+
CLR   @R2↑
```

3. After successfully completing a search command, EDIT positions the pointer immediately following the text object. Using a search command in combination with =J places the pointer in front of the text object, as follows:

```
*GTEST$=J$$
```

   This command combination places the pointer before TEST in the text buffer.

Find (nF)

The Find (nF) command starts at the current pointer location and searches the entire input file for the nth occurrence of the text string. If EDIT does not find the nth occurrence of the text string in the current buffer, it automatically performs a Next command and continues the search on the new text in the buffer. When the search is successful, EDIT leaves the pointer immediately following the nth occurrence of the text string.

If the search fails (that is, EDIT detects the end-of-file for the input file and does not find the nth occurrence of the text string), EDIT displays *?EDIT-F-Search failed*. In this instance, EDIT positions the pointer at the beginning of an empty text buffer. When you use the F command, EDIT deletes the contents of the buffer after writing it to the output file.

The syntax of the command is: **nFtext**

The argument n must be positive. If you omit it, EDIT assumes it to be 1.

You can use an F command to copy all remaining text from the input file to the output file by specifying a nonexistent text object. The Find command functions like a combination of the Get and Next commands.

Example

The following command searches the entire input file for the second occurrence of the text string MOVB6(R1),@R2. EDIT places the pointer following the text string. EDIT writes the contents of each unsuccessfully searched buffer to the output file:

```
*2FMOVB  6(R1),@R2$$
```

Position (nP)

The Position (nP) command is identical to the Find (F) command with one exception. The F command transfers the contents of the text buffer to the output file as each page is unsuccessfully searched, but the P command deletes the buffer contents after it is searched without writing text to the output file.

The syntax of the command is: **nPtext**

The argument n must be positive. If you omit it, EDIT assumes it to be 1.

The nP command searches each page of the input file for the nth occurrence of the text object, starting at the pointer and ending with the last character in the buffer. If EDIT finds the nth occurrence, it positions the pointer following the text object, deletes all pages preceding the one containing the text object, and positions the page containing the text object in the buffer.

If the search is unsuccessful, EDIT clears the buffer but does not transfer any text to the output file. EDIT positions the pointer at the beginning of an empty text buffer.

The P command is a combination of the Get, Delete, and Read commands; it is most useful as a means of placing the pointer in the input file. For example, if your aim in the editing session is to create a new file from the second half of the input file, a P search saves time.

Examples

1. This command searches the input file for the first occurrence of the text object, 3. EDIT positions the pointer after the text object:

   ```
   *P3$$
   ```

2. The next command lists on the terminal the current line up to the pointer:

   ```
   *0L$$
   INPUT FILE PAGE 3
   ```

Text-Listing Commands

Two EDIT commands display lines of text on the terminal:

- The List command displays lines of text as they appear in the buffer (nL).
- The Verify command displays the entire line in which the pointer is located (V).

List (nL)

The List (nL) command displays at the terminal lines of text as they appear in the buffer. An argument preceding the L command indicates the portion of text to print. For example, the command, 2L, displays on the terminal the text beginning at the pointer and ending with the second end-of-line character. The pointer is not altered by the L command.

The syntax of the command is: **nL**

where:

n	displays at the terminal n lines beginning at the pointer and ending with the nth end-of-line character.
-n	displays all characters beginning with the first character on the -nth line and terminating at the pointer.
0	displays the current line up to the pointer. Use this command to locate the pointer within a line.
/	displays the text between the pointer and the end of the buffer.

Examples

1. This command displays all characters starting at the beginning of the second preceding line and ending at the pointer:

```
*-2L$$
```

2. This command displays all characters beginning at the pointer and terminating at the fourth return and line-feed combination:

```
*4L$$
```

3. Assume the pointer location is:

```
MOVB     5(R1),@R2
ADD↑     R1,(R2)+
```

The next command displays the previous one and one-half lines of code up to the pointer:

```
*-1L$$
```

When the preceding command is executed, the terminal output looks like this:

```
MOVB  5(R1),@R2
ADD↑
```

Text-Listing Commands

Verify (V)

The Verify (V) command displays at the terminal the entire line in which the pointer is located. It provides a ready means of determining the location of the pointer after a search completes and before you give any editing commands. (The V command combines the two commands 0LL.)

You can also type the V command after an editing command to allow proofreading of the results.

The syntax of the command is: **V**

No arguments are allowed with the V command. The location of the pointer does not change.

Text-Modification Commands

You can use the following commands to:

- Insert text (I).
- Delete characters (nD).
- Remove lines (nK).
- Change characters (nC).
- Change lines (nX).

Insert (I)

The Insert (I) command is the basic command for inserting text. EDIT inserts the text you supply at the location of the pointer and then places the pointer after the last character of the new text.

The syntax of the command is: **Itext**

No arguments are allowed with the insert command. The text string is limited only by the size of the text buffer and the space available. All characters are legal, except ESCAPE which terminates the text string.

NOTE

If you forget to type the I command, the editor interprets the text as commands.

EDIT automatically protects the text buffer from overflowing during an insert. If the I command is the first command in a multiple command line, EDIT ensures that there will be enough space for the insert to be executed at least once. If repetition of the command exceeds the available memory, an error message displays.

Example

The following example illustrates the I command:

```
*IMOV   BUFF,R2
MOV     LINE,R1
MOVB    -1(R2),R0$$
*
```

This command inserts the text at the current location of the pointer and leaves the pointer positioned after R0.

Digital recommends that you insert large amounts of text into the file in small sections rather than all at once. This way, you are less vulnerable to loss of time and effort due to machine failure or human error. This is the recommended procedure for inserting large amounts of text:

1. Open the file with the EB command.
2. Insert or edit a few pages of text.

3. Insert a unique text string (likemrkplc) to mark your place.
4. Use the Exit command to preserve the work you have done so far.
5. Start again, using the F command to search for the unique string you used to mark your place.
6. Delete your marker and continue editing.

Delete (nD)

The Delete (nD) command is a character-oriented command that deletes n characters in the text buffer, beginning at the current location of the pointer. If you do not specify n, EDIT deletes the character immediately following the pointer. On completion of the D command, EDIT positions the pointer immediately before the first character following the deleted text.

The syntax of the command is: **nD**

where:

n	deletes n characters following the pointer. Places the pointer before the first character following the deleted text.
-n	deletes n characters preceding the pointer. Places the pointer before the first character following the deleted text.
0	deletes the current line up to the pointer. The position of the pointer does not change (equivalent to 0K).
/	deletes the text between the pointer and the end of the buffer. Positions the pointer at the end of the buffer (equivalent to /K).
=	Deletes -n characters, where n equals the length of the last text argument used.

Examples

1. This command deletes the two characters immediately preceding the pointer:

```
*-2D$$
```

2. This command string deletes the text string MOVR1 (=D in combination with a search command deletes the indicated text string):

```
*B$FMOVR1$=D$$
```

3. Assume the text buffer contains the following:

```
ADD  R1,(R2)+
CLR  ↑ @R2
```

The following command deletes the current line up to the pointer:

```
*0D$$
```

Once the preceding command is executed, the buffer contains:

```
ADD     R1,(R2)+
↑ @R2
```

Kill (nK)

The Kill (nK) command removes n lines of text (including the return and line-feed characters) from the page buffer, beginning at the pointer and ending with the nth end-of-line. EDIT places the pointer at the beginning of the line following the deleted text.

The syntax of the command is: **nK**

where:

n	removes the character string (including the return and line-feed combination) beginning at the pointer and ending at the nth end-of-line.
-n	removes the current line up to the pointer and n full lines preceding the current line. Thus, if the pointer is at the center of a line, the modifier -1 deletes one and one-half lines preceding it.
0	removes the current line up to the pointer (equivalent to 0D).
/	removes the characters beginning at the pointer and ending with the last line in the text buffer (equivalent to /D).

Examples

1. This command deletes lines starting at the current location of the pointer and ending at the second return and line-feed combination:

```
*2K$$
```

2. Assume the text buffer contains the following:

```
ADD     R1,(R2)+
CLR↑    @R2
MOVB    6(R1),@R2
```

This command removes the characters beginning at the pointer and ending with the last line in the text buffer:

```
*/K$$
```

Once the preceding command is executed, the buffer contains:

```
ADD     R1,(R2)+
CLR↑
```

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is character-oriented.

Change (nC)

The Change (nC) command changes a specific number of characters preceding or following the pointer. A C command is equivalent to a Delete command followed by an Insert command. You must insert a text object following the nC command.

The syntax of the command is: **nCtext**

where:

n	replaces n characters following the pointer with the specified text. Places the pointer after the inserted text.
-n	replaces n characters preceding the pointer with the specified text. Places the pointer after the inserted text.
0	replaces the current line up to the pointer with the specified text. Places the pointer after the inserted text (equivalent to 0X).
/	replaces the text beginning at the pointer and ending with the last character in the buffer. Places the pointer after the inserted text (equivalent to /X).
=	replaces -n characters with the indicated text string, where n represents the length of the last text argument used.

The size of the text is limited only by the size of the text buffer and the space available. All characters are legal except ESCAPE which terminates the text string.

If the C command is to be executed more than once (that is, it is enclosed in angle brackets) and if there is enough space available for the command to be entered, it will be executed at least once (provided it appears first in the command string). If repetition of the command exceeds the available memory, an error message displays.

Examples

1. This command replaces the five characters to the right of the pointer with VECT:

```
*5CVECT$$
```

2. Assume the text buffer contains the following:

```
CLR     @R2
MOV↑    5(R1),@R2
```

   The next command replaces the current line up to the pointer with the specified text:

```
*0CADDB$$
```

   After the preceding command is executed, the buffer contains:

```
CLR       @R2
ADDB↑     5(R1),@R2
```

3. You can use =C with a Get command to replace a specific text string. Here is an example:

```
*GFIFTY:$=CFIVE:$
```

   This command finds the text string FIFTY: and replaces it with FIVE:.

Exchange (nX)

The Exchange (nX) command is similar to the change command, except that it changes lines of text instead of a specific number of characters. The nX command is identical to an nK command followed by an Insert command.

The syntax of the command is: **nXtext**

where:

n	replaces n lines, including the return and line feed characters, following the pointer. Places the pointer after the inserted text.
-n	replaces n lines, including the return and line-feed characters, preceding the pointer. Positions the pointer after the inserted text.
0	replaces the current line up to the pointer with the specified text. Positions the pointer after the inserted text (equivalent to 0C).
/	replaces the text beginning at the pointer and ending with the last character in the buffer with the specified text (equivalent to /C). Positions the pointer after the inserted text.

All characters are legal in the text string except ESCAPE which terminates the text.

If the X command is enclosed within angle brackets to allow more than one execution, and if there is enough memory space available for the X command to be entered, EDIT executes it at least once (provided it is first in the command string). If repetition of the command exceeds the available memory, an error message displays.

Example

This command exchanges the two lines to the right of the pointer with the text string:

```
*2XADD  R1,(R2)+
CLR     @R2
$$
*
```

Utility Commands

During an editing session, you can:

- Store text in external buffers (nS).
- Restore text from the external buffer back into your text buffer (U).
- Insert a command string into the EDIT macro buffer (M).
- Execute a command string stored in the macro buffer (nEM).
- Display the version number of the editor (EV).
- Enable both uppercase and lowercase letters (EL).
- Enable only uppercase letters (the default) (EU).

Save (nS)

The Save (nS) command lets you store text in an external buffer called a save buffer (described previously in the Text-Listing Commands section, and subsequently insert it in several places in the text.

The syntax of the command is: **nS**

The Save command does the following:

- Copies n lines, beginning at the pointer, into the save buffer.
- Operates only in the forward direction; therefore, you cannot use a negative argument.
- Destroys any previous contents of the save buffer; however, EDIT does not change the location of the pointer or the contents of the text buffer.

If you specify more characters than the save buffer can hold, EDIT displays the error message *?EDIT-F-Insufficient memory*. None of the specified text is saved.

Example

Assume the text buffer contains the following assembly language subroutine:

```
;subroutine MSGTYP
;When called, expects R0 to point to an
;ASCII message that ends in a zero byte,
;types that message on the user terminal

MSGTYP:  TSTB    (R0)           ;Done?
         BEQ     MDONE          ;Yes-Return
MLOOP:   TSTB    @#177564       ;No-Is terminal ready?
         BPL     MLOOP          ;No-Wait
         MOVB    (R0)+,@#177566 ;Yes Print character
         BR      MSGTYP         ;Loop
MDONE:   RTS     PC             ;Return
```

The following command stores the entire subroutine in the save buffer (assuming the pointer is at the beginning of the buffer):

```
*12S$$
```

You can insert the contents of the save buffer into a program whenever you choose by using the Unsave command.

Unsave (U)

The Unsave (U) command inserts the entire contents of the save buffer into the text buffer at the pointer and leaves the pointer positioned following the inserted text. You can use the U command to move blocks of text or to insert the same block of text in several places.

The syntax for using the U command is: **[0]U**

where:

U	inserts the contents of the save buffer into the text buffer.
0U	clears the save buffer and reclaims the area for text.

The contents of the save buffer are not destroyed by the U command (only by the 0U command) and can be unsaved as many times as desired. If the Unsave command causes an overflow of the text buffer, the *?EDIT-F-Insufficient memory* error message displays, and the command does not execute.

Example

This command inserts the contents of the save buffer into the text buffer:

```
*U$$
```

Macro (M)

The Macro (M) command inserts a command string, called a macro, into the EDIT macro buffer.

The M commands and their functions are:

M/command string/	stores the command string in the macro buffer.
0M or **M//**	clears the macro buffer and reclaims the area for text.

The slash (/) represents the delimiter character. The delimiter is always the first character following the M command, and can be any character that does not appear in the macro command string itself.

Starting with the character following the delimiter, EDIT places the command string characters into its internal macro buffer until the delimiter is encountered again. At this point, EDIT returns to command mode. The Macro command does not execute the command string; it merely stores the command string so that the Execute Macro (EM) command can execute later. The Macro command does not affect the contents of the text or save buffers.

All characters except the delimiter are valid macro command string characters, including single ESCAPEs to terminate text commands. All commands, except the M and EM commands, are valid in a command string macro.

In addition to using the 0M command, you can type the M command immediately followed by two identical characters (assumed to be delimiters) and two ESCAPE characters to clear the macro buffer.

Examples

1. This command clears the macro buffer:

```
*M//$$
```

2. This command stores a macro to change R0 to R1:

```
*M/GR0$-C1/$$
```

NOTE

Choose infrequently used characters as macro delimiters; choosing frequently used characters can lead to errors. For example:

```
*M GMOV R0$=CADD R1$ $$
?EDIT-F-No file open for input
```

In this case, it was intended that the macro be GMOV R0$=CADD R1$, but since the delimiter character (the character following the M) is a space, the space following MOV is used as the second delimiter, terminating the macro. EDIT then returns an error when it interprets the R as a Read command.

Execute Macro (nEM)

The Execute Macro (nEM) command executes a command string previously stored in the macro buffer by the M command.

The syntax of the command is: **nEM**

The argument n must be positive. The macro is executed n times and then returns control to the next command in the original command string.

Examples

1. This command sequence stores a command in the macro buffer and then executes that command. EDIT displays an error message when it reaches the end of the buffer. (This macro changes all occurrences of R0 in the text buffer to R1.):

```
*M/BGR0$-C1$/$$
*B1000EM$$
?EDIT-F-Search failed
*
```

2. This command inserts MOV PC,R1 into the text buffer and then executes the command in the macro buffer twice before inserting CLR @R2 into the text buffer:

```
*IMOV PC,R1$2EMICLR@R2$$
*
```

Edit Version (EV)

The Edit Version (EV) command displays the version number of the editor in use on the terminal.

The syntax of the command is: **EV**

Example

This command displays the running version of EDIT:

```
*EV$$
V05.00
*
```

Lowercase (EL) and Uppercase (EU) Commands

If you have a terminal that has both uppercase and lowercase characters as part of your hardware configuration, you can take advantage of two editing commands, Edit Lower (EL) and Edit Upper (EU).

When the editor is started with the EDIT command, uppercase mode is assumed; that is, all characters you type are automatically translated to uppercase. To allow processing of both uppercase and lowercase characters, enter the Edit Lower command.

Examples

1. After executing the EL command, you enable the editor to accept and echo (and display) uppercase and lowercase characters received from the keyboard:

```
*EL$$
*i You can enter text and commands in UPPER and lower case.$$
*
```

2. To return to uppercase mode, use the Edit Upper command:

```
*EU$$
```

   Control also reverts to uppercase mode on exit from the editor (with EX or CTRL/C).

3. Note that when you issue an EL command, you can enter EDIT commands in either uppercase or lowercase. Thus, the following two commands are equivalent:

```
*GTEXT$=Cnew text$V$$
*gTEXT$=cnew text$v$$
```

The editor automatically translates (internally) all commands to uppercase without reference to EL or EU.

NOTE

When you use EDIT in EL mode, make sure that text arguments you specify in search commands have the proper case. The command GTeXt$, for example, will not match TEXT, text, or any combination other than TeXt.

EDIT Commands Summary

Command	Name	Function
nA	Advance	Advances the location pointer a specified number of lines forward.
B	Beginning	Places the location pointer at the beginning of the buffer.
nC	Change	Changes characters.
nD	Delete	Deletes characters.
EB	Edit Backup	Opens an existing file for editing and creates a backup version of it.
EF	End File	Closes and opens the output file.
EL	Edit Lower	Enables both uppercase and lowercase letters.
nEM	Execute Macro	Executes a command string stored in the macro buffer.
ER	Edit Read	Opens an existing file for input and prepares it for editing.
EU	Edit Upper	Enables only uppercase letters (the default).
EV	Edit Version	Displays the version number of the editor.
EW	Edit Write	Opens a file for output of newly created or edited text.
EX	Exit	Terminates the editing session.
nF	Find	Locates the *n*th occurrence of a specified text string in the input file while transferring the buffer contents to the output file as each page is unsuccessfully searched.
nG	Get	Locates the *n*th occurrence of a specified text string in the text buffer.
I	Insert	Inserts text.
nJ	Jump	Places the location pointer a specified number of characters forward.
nK	Kill	Removes lines.
nL	List	Displays lines of text as they appear in the buffer.
M	Macro	Inserts a command string into the EDIT macro buffer.
nN	Next	Copies lines of text from the text buffer to the output file, while deleting the text from the buffer and reading the next page of the input file into the buffer.
nP	Position	Locates the *n*th occurrence of a specified text string in the input file while deleting the buffer contents as each page is unsuccessfully searched.

EDIT Commands Summary

Command	Name	Function
R	Read	Reads text from an input file into the buffer.
nS	save	Stores text in external buffers.
U	Restore	Restores text from the external buffer back into your text buffer.
V	Verify	Displays the entire line in which the pointer is located.
nW	Write	Copies lines of text from the text buffer to the output file, while not altering the buffer.
nX	Exchange	Changes lines.

EDIT Error Conditions

The editor displays an error message whenever it detects an error. EDIT checks for three general types of error conditions:

Syntax errors
Execution errors
Macro-execution errors

The following list describes the error message form for each type of error condition:

- Before it executes any commands, EDIT first scans the entire command string for errors in command syntax, such as illegal arguments or an illegal combination of commands. If the editor finds an error of this type, it displays a message of this form:

  ```
  ?EDIT-F-Message;no command(s) executed
  ```

  You should retype the command.

- If a command string is syntactically correct, EDIT begins execution. Execution errors, such as buffer overflow or input and output errors, can still occur. In this case, EDIT displays a message of the form:

  ```
  ?EDIT-F-Message
  ```

  EDIT executes all commands preceding the one in error. It does not execute the command in error or any commands that follow it.

- When an error occurs during execution of a macro, EDIT displays a message of the form:

  ```
  ?EDIT-F-Message in macro; no command(s) executed
  ```

  or

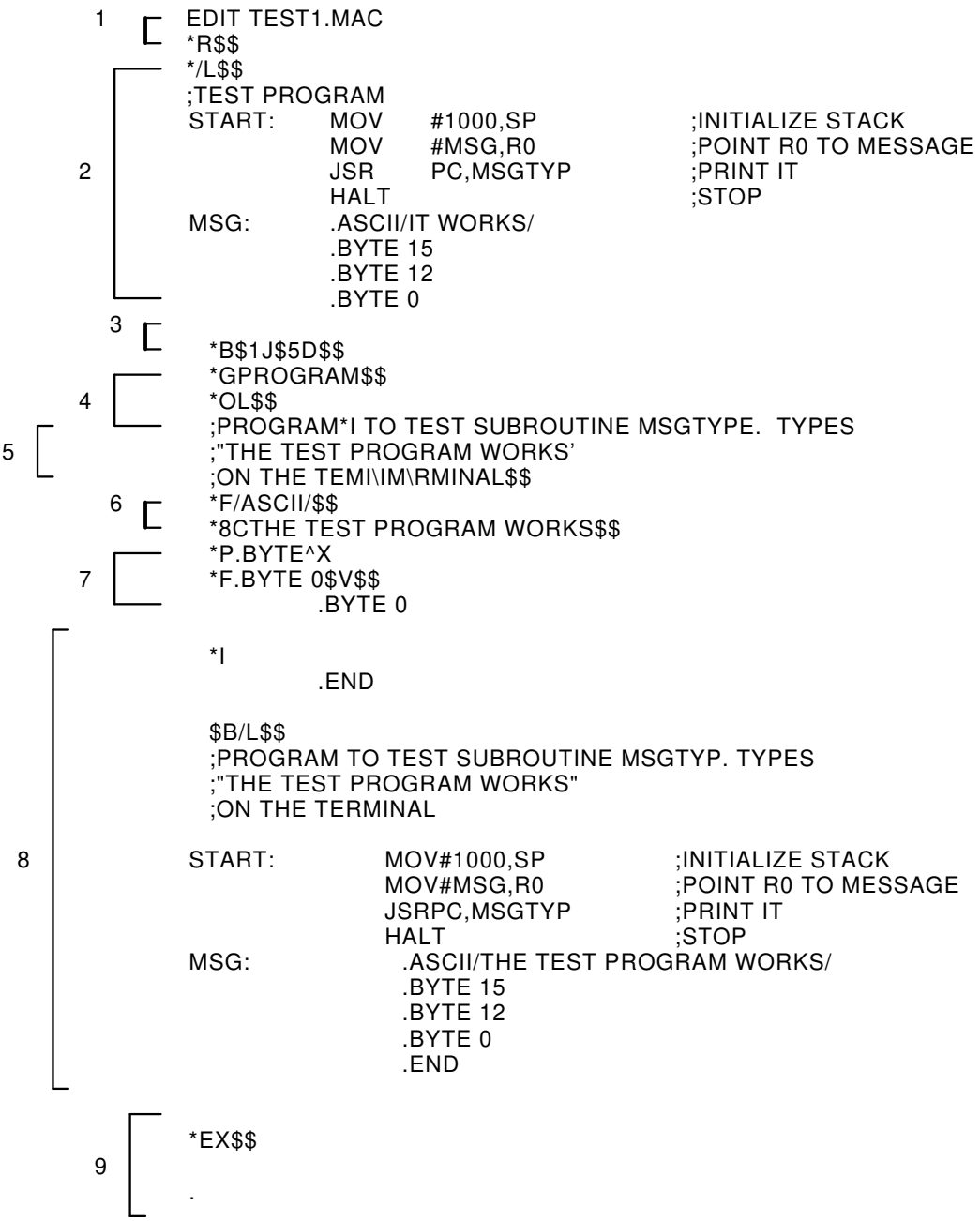
  ```
  ?EDIT-F-Message in macro
  ```

Most errors are syntax errors. These are usually easy to correct before execution.

The *RT–11 System Message Manual* contains a complete list of the EDIT error messages, along with recommended corrective action for each error.

Example Editing Session

The following example illustrates the use of EDIT commands to change a program stored on the DK device:



The following list explains the numbered sections in the example:

1. Calls the EDIT program and displays *. The input file is TEST1.MAC; the output file is TEST2.MAC. Reads the first page of input into the buffer.
2. Lists the buffer contents.
3. Places the pointer at the beginning of the buffer. Advances the pointer one character (past the ;) and deletes the TEST.
4. Positions the pointer after PROGRAM and verifies the position by listing up to the pointer.
5. Inserts text. Uses DELETE to correct typing error.
6. Searches for .ASCII/ and changes ITWORKS to THE TEST PROGRAM WORKS.
7. Types CTRL/X to cancel the P command. Searches for .BYTE 0 and verifies the location of the pointer with the V command.
8. Inserts text. Returns the pointer to the beginning of the buffer and lists the entire contents of the buffer.
9. Closes the input and output files after copying the current text buffer as well as the rest of the input file into the output file. EDIT returns control to the monitor.

Chapter 9
The Error-Logging Package

The Error-Logging Package (EL, ELINIT, ERRLOG, and ERROUT) is a group of programs that monitor the hardware reliability of your computer system.

Forms of the Error Logger

The primary error-logging component, called the Error Logger, gathers a log of error-related information and stores it. This component varies, depending on the type of monitor you chose for your system. Table 9–1 explains which Error Logger goes with which monitor and how the Error Logger runs under that monitor.

Table 9–1: Forms of the Error Logger

Error Logger	Monitor	Form
EL.SYS	RT11SB	Handler
ELX.SYS	RT11XB	Handler
ELX.SYS	RT11ZB	Handler
ERRLOG.REL	RT11FB	Foreground job
ERRLOG.REL	RT11XM	Foreground or system job
ERRLOG.REL	RT11ZM	Foreground or system job

Generating a System with Error Logging

Error logging is available only as a special feature; that is, you must perform a system generation to create the error-logging files and enable error logging. It is available under all monitors.

Although you can select (T)MSCP error-logging support regardless of the monitor environment, you are urged to select it only when running a mapped monitor. If you run the Error Logger under an unmapped monitor, you are using valuable low memory, reducing the memory available to run other jobs.

Digital distributes the file XMEL.ANS, which is a system-generation answer file to produce the distributed XM monitor with the addition of error-logging support for both non-MSCP devices and (T)MSCP devices.

Error-Logging Functions

The error-logging components work together to:

- Gather (from the appropriate device handlers) a statistical record of all I/O operations that occur on any of the following devices:

DD	DX
DL	DY
DM	DZ
MSCP DU	TMSCP MU
DW	RK

- Detect (from the monitor) and record memory-parity or cache errors and any errors that occur during I/O operations.
- Keep four counts of successful I/O operations, including successful .SPFUN operations:
  - — Read successes
  - — Write successes
  - — Motion successes
  - — Other (default)

  .SPFUN operations affect all the counts, but the last two apply only to .SPFUNS. Special directory operations are always logged as *other*. For an explanation of the *motion* and *other* counts, see *RT–11 System Internals Manual* for information on the .DRSPF macro.
- Store the information the Error Logger gathers in a file (if the Error Logger is running under a multi-job monitor—FB, XM, or ZM) or in an internal buffer (if the Error Logger is running under a single-job monitor—SB, XB, or ZB).
- Format and produce at intervals you determine individual and/or summary reports on some or all of the preceding types of operations.

  Error-logging reports are useful for maintaining the hardware on which RT–11 runs. Problems such as line noise, static discharges, or inherently error-prone media can cause recoverable errors on systems that are otherwise functioning normally. By studying error-logging reports, you can learn to distinguish these errors from those that might be symptoms of an impending device failure. Also, some recoverable errors that are insignificant in themselves might be related to other more serious errors; their effects might not be immediately apparent to you. Information contained in the reports about each error and about the status of the system when the error occurred may alert you to a previously unforeseen hardware problem.

Sometimes a device fails so quickly that you are unable to prevent it. In this case, you can determine the cause more quickly if a report is available that describes the errors that occurred immediately prior to the failure.

NOTE

Because the Error Logger can record data on each I/O transfer, thereby using additional computer time and memory, you may wish to use the Error Logger only when you experience difficulty with a device. Keeping a backup system volume on which the Error Logger is enabled makes this easy. You can also issue the command SET dd: NOSUCCES (dd represents the device mnemonic) before running the Error Logger. This command causes the device to call the Error Logger only when an I/O transfer fails. Successful I/O transfer statistics are not recorded. (Remember to reload the dd handler after issuing the SET dd: NOSUCCES command.)

- Fully support MSCP (DU) and TMSCP (MU) error logging under all monitors.

  (T)MSCP error logging can generate datagrams that pinpoint the reason for an error more precisely than previous RT–11 error-logging implementations. You can save yourself time and money replacing what you might have thought was a bad cable or disk by analyzing the (T)MSCP error reports and troubleshooting the correct cause of the problem.

  The mapped-monitor error-logging system for (T)MSCP devices minimizes the size increase of the low-memory part of the handler that is due to error logging.

Error-Logging Components

The Error Logger is sometimes called a *package* or *subsystem* since it consists of several components that work together.

Distributed Source Components

The following are the distributed source components for the Error Logger:

- EL.MAC
- ELINIT.MAC
- ELTASK.MAC
- ELCOPY.MAC
- ERROUT.OBJ
- ERROUT.SAV

When you generate an error-logging system, the error-logging components vary, depending on whether the generated system has a single- or multi-job monitor.

You need ERROUT.OBJ as a source file only if you want to include your own handlers in your error-logging system.

Error-Logging Components for Single-Job Monitors

When the Error Logger is used with a single-job monitor, it consists of the components described in Table 9–2. They are described according to their names, their sources, and the monitors with which you can run them.

Table 9–2: Error-Logging Components for Single-Job Monitors

Program	Source	Monitor
EL.SYS	EL.MAC	Single-job, unmapped monitor (RT11SB)
ELX.SYS	EL.MAC	Single-job, mapped monitors (RT11XB and RT11ZB)
ERROUT.SAV	Distribution kit	All monitors

EL.SYS and ELX.SYS are error-logging handlers. The Error Logger (EL) gathers I/O and error-related information in an internal buffer area (ERRLOG.DAT is not created). You can generate a report from the information in the internal buffer by calling ERROUT.SAV, a report generator.

Error-Logging Components for Multi-Job Monitors

When the Error Logger (ERRLOG) is used with a multi-job monitor, it consists of the components described in Table 9–3. They are described according to their names, their sources, and the monitors with which you can run them.

Table 9–3: Error-Logging Components for Multi-Job Monitors

Program	Source	Monitor
ERRLOG.REL	ELCOPY.MAC and ELTASK.MAC	Multi-job monitors (RT11FB, RT11XM, and RT11ZM)
ELINIT.SAV	ELINIT.MAC	Multi-job monitors (RT11FB, RT11XM, and RT11ZM)
ERROUT.SAV	Distribution kit	All monitors

In the case of a multi-job environment:

1. You run ERRLOG.REL as a foreground job under RT11FB or a system job under RT11XM or RT11ZM.
2. You enable error logging by running ELINIT.SAV as a background job. ELINIT creates and/or initializes a statistics file called ERRLOG.DAT. ERROUT.SAV reports error-log information to ERRLOG.DAT.
3. At any time you specify, you call ERROUT to create error-log reports from the information in ERRLOG.DAT. With ERROUT, you can display an error-log report on the terminal, send it to a printer, or place it in a file.

Descriptions of the Error-Logging Programs

EL.SYS or ELX.SYS

A pseudohandler used with a single-job monitor to gather information about errors that occur during I/O transfers. The device handlers detect success and error information as each I/O transfer occurs. The handlers communicate this information to EL.SYS (in an unmapped environment) or to ELX.SYS (in a mapped environment), which gathers the necessary statistics for an error report. EL.SYS/ELX.SYS stores these statistics in an internal buffer whose default size is 1 block.

You can change the size of the internal buffer by setting the conditional ERL$S (in SYCND.MAC) to n, where n is the number of blocks you want to reserve for the internal buffer. The variable n is interpreted as an octal number, unless you include a decimal point.

ERRLOG.REL

A foreground or system job that gathers information about I/O transfers and system errors. The device handlers detect success and error information as each I/O transfer occurs. The handlers communicate this information to ERRLOG.REL, which stores all the necessary statistics for an error report in an internal buffer. The buffer's contents are periodically transferred to ERRLOG.DAT, or whenever you request an error report. When you initiate error logging with a multi-job monitor, ERRLOG.REL instructs you to run ELINIT.

ELINIT

A background job under a multi-job monitor that creates and initializes the statistics file, ERRLOG.DAT, and maintains the file's size:

- ELINIT creates ERRLOG.DAT when you enable error logging.
- ELINIT initializes ERRLOG.DAT every time you have an error-logging session at the terminal after you have enabled error logging. You might want to initialize ERRLOG.DAT after you have created an error report or if there is no longer room in ERRLOG.DAT for more statistics.
- When you run ELINIT, it prompts you for the information it needs to maintain ERRLOG.DAT's size. By default, ELINIT allocates 100 decimal blocks for ERRLOG.DAT. Each time you run ELINIT, it displays a message that tells how many of those 100 blocks are filled. If ERRLOG.DAT fills to its limit, EL.SYS (or ELX.SYS) is unable to store more information in it. So that you can increase ERRLOG.DAT's size, ELINIT prompts you for a file-size change each time you run the program.
- If you bootstrap a monitor whose features differ from those of the monitor under which ERRLOG.DAT was created, ELINIT may display a message indicating that it must initialize ERRLOG.DAT to make the statistics it has been maintaining compatible with the new configuration. When this happens, ELINIT renames the ERRLOG.DAT it formerly maintained to ERRLOG.TMP

and creates a new ERRLOG.DAT. The Error Logger can still create a report from ERRLOG.TMP.

Note that you do not use ELINIT when you run the error logger with a single-job monitor. Instead, the Error Logger compiles statistics in an internal buffer area. When the internal buffer area fills to its limit, the Error Logger is unable to store more information in it. You can generate a report from the information in the internal buffer or purge the internal buffer at any time.

ERROUT

A report generator that runs as a background job. ERROUT creates a report from the statistics in the error-logger internal buffer area, or from ERRLOG.DAT, or from any file of that format. When you run ERROUT, you can direct the program to list the error report at the terminal or on a printer, or to create a file for the error report. You can also indicate whether you want a detailed report on each error that occurred or simply a summary report.

Note that Digital recommends you use a device other than the DU device on the same controller to log DU errors in the file ERRLOG.DAT. Logging errors to ERRLOG.DAT on the same DU device and controller that contains your working system and/or application imposes a burden on that device.

You can log errors to any file-structured or magtape device, although any magtape device must contain the FSM (as distributed). If you plan to use a DU device on a different controller, Digital recommends you use the technique described in *RT–11 Device Handlers Manual* to create a second DU handler, BU, and use BU to contain ERRLOG.DAT. The second handler need not support error logging but must operate through a different MSCP controller than the first (DU) handler.

Diagrams of How Error Logging Works

The following two diagrams show how error logging works. The first diagram shows error logging under a single-job monitor, while the second diagram shows error logging under a multi-job monitor.

DIAGRAM 1

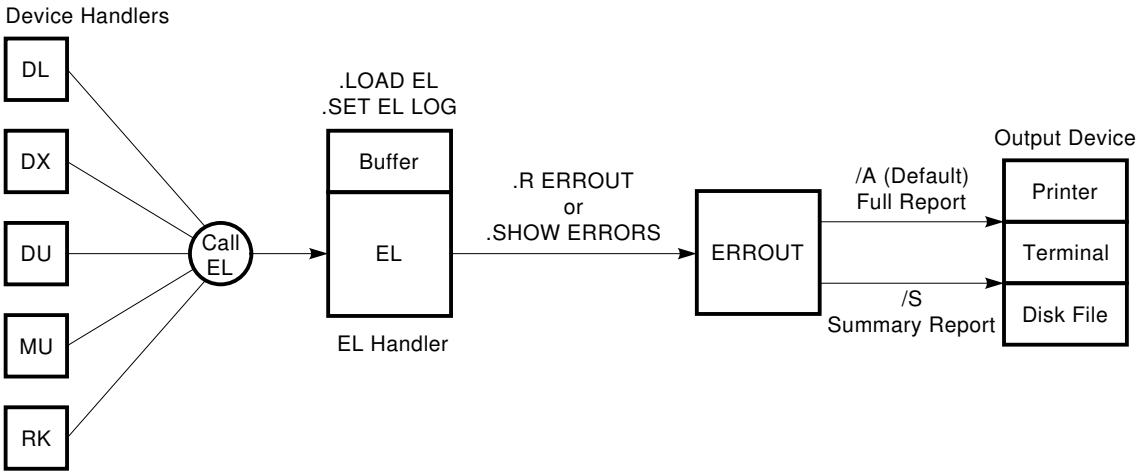
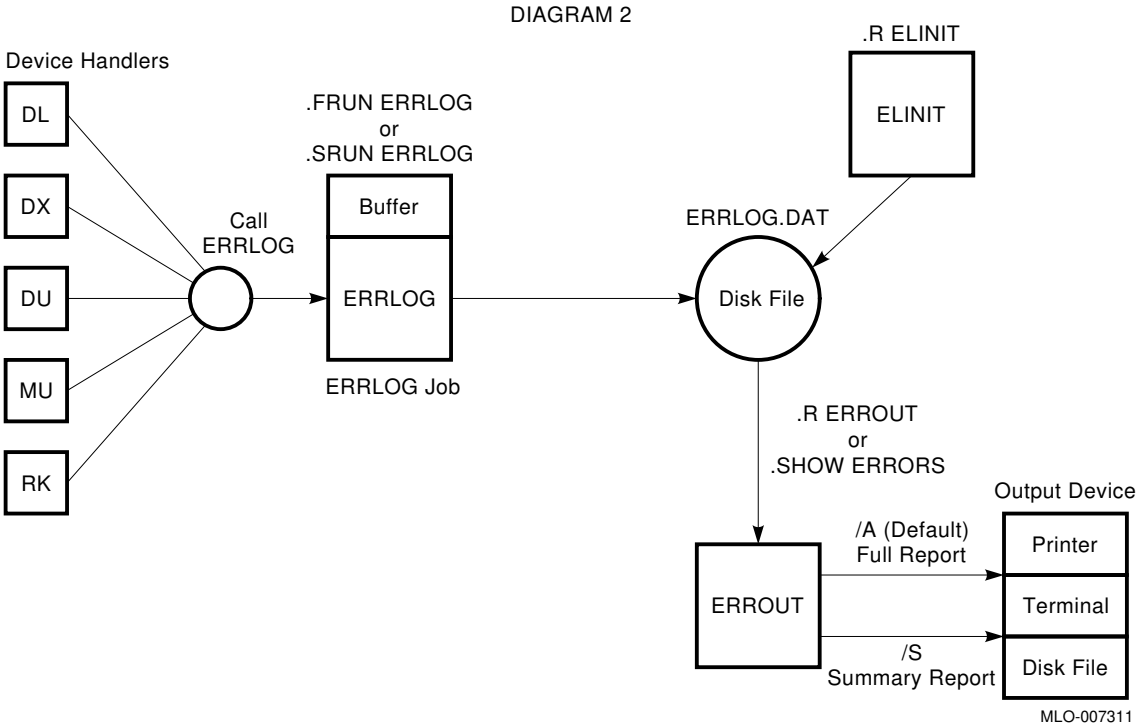


DIAGRAM 2



Using the Error Logger with a Single-Job Monitor

When using the Error Logger with a single-job monitor, you should know how to:

- Load and start the Error Logger
- Clear the Error Logger's buffer
- Generate reports
- Suspend error logging
- Resume error logging
- Terminate error logging

The following sections explain each of the preceding procedures.

Loading and Starting the Error Logger

First, to run the Error Logger with a single-job monitor, load the Error Logger pseudohandler with the DCL command:

LOAD EL

Second, to enable error logging, issue the DCL command:

SET EL LOG

When you issue this command, the Error Logger begins to gather I/O transfer and error information in an internal buffer. The Error Logger also gathers statistics on the number of successful I/O transfers but does not create detailed records about successful transfers in the internal buffer.

Clearing the Error Logger's Buffer and Generating a Report

The Error Logger creates detailed records only for errors; these records contain such information as the device involved, when the error occurred, register contents, and number of retries. If the buffer becomes full, EL continues to compile I/O transfer statistics but writes no further detailed records to the internal buffer. When this occurs, the Error Logger displays the following message:

```
?EL-W-Buffer is full, logging suspended
```

To clear the contents of the internal buffer when it becomes full, or at any other time, issue the DCL command:

SET EL PURGE

This command clears only the detailed records on errors stored in the internal buffer; the I/O statistics are retained.

To generate a report when the buffer is full or at any other time, see the Displaying, Printing, or Saving Error-Log Reports section.

Suspending, Resuming, and Disabling Error Logging

- To suspend error logging, issue the DCL command:

  SET EL NOLOG

- To resume error logging after you have suspended it, issue the DCL command:

  SET EL LOG

- To disable error logging and unload the EL pseudohandler when you are through using the Error Logger, issue the DCL command:

  UNLOAD EL

  This command clears the EL internal buffer area and all I/O statistics. If you want to save the contents of the internal buffer in a file, copy it to a file before you unload EL.

- To save the internal buffer contents, issue a DCL command with the following syntax:

  COPY EL: dev:filnam.typ

Using the Error Logger with a Multi-Job Monitor

Note that you cannot run more than one error logger, and the job name for that single Error Logger must be ERRLOG.

To use the Error Logger with a multi-job monitor, you should know how to:

- Call the Error Logger.
- Terminate the Error Logger.
- Enable and set up the Error Logger according to your specifications by running ELINIT.
- List, print, or save error-log reports by running ERROUT.

The following sections explain each of these procedures.

Calling the Error Logger with a Multi-Job Monitor

With a multi-job monitor, the Error Logger runs only as a foreground or system job.

- To run the Error Logger as a foreground job, issue the DCL command:

  FRUN ERRLOG

- To run the Error Logger as a system job, issue the DCL command:

  SRUN ERRLOG

  The Error Logger returns with a prompt, telling you how to initiate the error-logging process:

  ```
  ?ERRLOG-I-To initiate Error Logging, RUN ELINIT
  ```

Terminating the Error Logger with a Multi-Job Monitor

To terminate the Error Logger:

- If it is running as a foreground job, type:
  1. CTRL/F
  2. CTRL/C CTRL/C
- If it is running as a system job, type:
  1. CTRL/X
  2. ERRLOG (as the system job you want to terminate)
  3. CTRL/C CTRL/C

Running ELINIT to Enable the Error Logger

After you issue the command to RUN ELINIT, ELINIT displays the prompt:

```
What is the name of the device for the ERRLOG.DAT file <SY>?
```

This prompt asks you to specify to which device you want the statistics file ERRLOG.DAT written. Type only RETURN in response to the prompt if you want ELINIT to write ERRLOG.DAT to the system device. Otherwise, specify the device you want.

ELINIT then displays a message indicating how many blocks allocated to ERRLOG.DAT are in use. This message is followed by a prompt asking you if you want ELINIT to initialize ERRLOG.DAT. The following is the format for the prompt, where *xx* represents the number of blocks in use, and *yy* represents the total number of available blocks:

```
xx blocks currently in use of yy possible total in ERRLOG.DAT file

Do you want to zero the ERRLOG.DAT file and re-initialize (YES/NO)
<NO>?
```

Type YES followed by RETURN if you want ELINIT to initialize ERRLOG.DAT. When ELINIT initializes ERRLOG.DAT, it does not create a backup file for the statistics that were present prior to initialization.

Press RETURN or type NO followed by RETURN if you want ELINIT to retain the statistics already compiled in ERRLOG.DAT.

ELINIT proceeds by issuing the following prompt, asking you to indicate the number of blocks you want ELINIT to allocate to ERRLOG.DAT:

```
How many blocks for the ERRLOG.DAT file <nnn>?
```

The variable *nnn* represents the default size of 100, or the size of the current ERRLOG.DAT file. Press RETURN if you want ERRLOG.DAT's file size to remain at the size indicated. If you want the file to be a different size, you can specify the number of blocks you want the file to have, followed by RETURN. ERRLOG.DAT must be larger than one block and can be as large as the available space permits on the device in which it resides.

NOTE

Because of a rearrangement of your RT–11 configuration or bad header information in ERRLOG.DAT, it may be necessary for ELINIT to initialize ERRLOG.DAT even if you do not want it to. In this case, ELINIT automatically renames the current ERRLOG.DAT to ERRLOG.TMP, displays a message indicating it has done so, and returns the prompt:

```
How many blocks for the ERRLOG.DAT file <100>?
```

After you have responded to the file-size prompt, ELINIT displays the following message:

```
RT-11 V5.6 ERROR LOGGING INITIATED
```

After the Error Logger has displayed that last message, you can proceed.

Displaying, Printing, or Saving Error-Log Reports

The report generator ERROUT creates a report from the information compiled in the file ERRLOG.DAT or in EL's internal buffer. You can instruct ERROUT to generate a report either indirectly, by typing the SHOW ERRORS command, or directly by running ERROUT. See the description of the SHOW ERRORS command in the *RT–11 Commands Manual* for more information on this command. See also the DCL Equivalents of ERROUT Operations section in this chapter. To call ERROUT directly from the system device, issue the DCL command:

RUN ERROUT

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to enter a command string according to the following general syntax:

[output-filespec=][input-filespec]/option

where:

output-filespec specifies the device to which you want ERROUT to type the report. If you do not specify an output device, ERROUT displays the report at the terminal. If you specify a file name, ERROUT writes the error-log report to that file. If you specify LP:, ERROUT writes the report to the printer.

input-filespec specifies ERRLOG.DAT or any file of the error logger statistics file format. (Thus, you can rename ERRLOG.DAT at any time and save it for later report formatting.) If you do not specify an input file, ERROUT assumes ERRLOG.DAT when running under a multi-job monitor, or EL.SYS's internal buffer area when running under a single-job monitor.

option specifies one of the options listed in Table 9–4.

ERROUT Options for Displaying Error-Log Reports

Table 9–4 lists the options available to display an error-log report.

Table 9–4: ERROUT Options

/A	Creates a report on each error in addition to a summary report of the errors and I/O transfers that occurred with each device.
/F:date	Creates an error report for errors logged from the date you specify. Specify the date in the form: **dd:mmm:yy** where: **dd** specifies the two-digit day. **mmm** specifies the first three letters of the month. **yy** specifies the last two digits of the year. ERROUT interprets the date you enter as octal; use a decimal point with the day and year to indicate the date is in decimal. If you do not use /F:date, ERROUT creates a report starting with the first error logged in the work file.
/S	Creates only a summary report of the errors and I/O transfers that occurred with each device.
/T:date	Creates an error report for errors logged up to the date you specify. Specify the date, using the same format as with the /F:date option. If you do not use /T:date, ERROUT creates a report that includes the last error logged in the work file.

If you press RETURN only in response to the CSI asterisk, ERROUT displays a full report from ERRLOG.DAT at the terminal.

DCL Equivalents of ERROUT Operations

Table 9–5 lists the DCL SHOW command options that are equivalent to ERROUT display operations.

The first part of the table lists that part of the ERROUT command syntax that is equivalent to a SHOW option. The rest of the table alphabetically lists all the ERROUT options having DCL SHOW option equivalents.

Table 9–5: DCL Equivalents of ERROUT Operations

ERROUT Utility Syntax/Option	SHOW Command Option
=filespec	/FILE:filespec
filespec=	/OUTPUT:filespec
LP:=	/PRINTER
TT:= (default)	/TERMINAL (default)
/A (default)	/ALL (default)
/F	/FROM[:date]
/S	/SUMMARY
/T	/TO[:date]

The DCL SHOW ERRORS command is equivalent to running ERROUT without options.

Non-MSCP Storage-Device Error-Log Report

When a device handler encounters an error during an I/O transfer, it automatically retries that transfer as many as eight times (the actual number of times a handler retries an unsuccessful transfer depends on the particular device handler and on the value you specify for n with the SET dd: RETRY=N command). Regardless of the number of retries, each unsuccessful transfer will be recorded as only one entry in the error report, unless the registers change during the retries. In that case, the Error Logger creates a report for each retry.

An example of a storage-device error report follows. This example is a report of the second attempt for a read operation on an RX02 double-density diskette. For ease of reference, each line in this example report is numbered (although lines in the actual report are not numbered). Following the report is a line-by-line analysis of it.

Example Storage-Device Error-Log Report

```
1  ****************************************************************
2  DISK DEVICE ERROR
3  LOGGED 8-OCT-90 16:12:45
4  ****************************************************************
5
6  UNIT IDENTIFICATION
7          PHYSICAL UNIT NUMBER                                 000001
8          TYPE                                               RX211/RX02
9
10 SOFTWARE STATUS INFORMATION:
11         MAXIMUM RETRIES       8.
12         REMAINING RETRIES     6.
13         OCCURRENCES OF THIS ERROR WITH IDENTICAL REGISTERS  2.
14
15 DEVICE INFORMATION
16         REGISTERS:
17         RX2CS             114560
18         RX2DB             010400
19         RX2ES             000120
20
21         ACTIVE FUNCTION                  READ
22         BLOCK                          000001
23         PHYSICAL BUFFER ADDRESS START  003734
24         TRANSFER SIZE IN BYTES            512.
```

Analysis of the Example Storage-Device Error-Log Report

The following is a line-by-line analysis of the preceding report.

Line	Explanation
1–4	Report header. Includes the date and time error was logged.
6–8	Unit identification. Identifies the drive number, the device controller, and the storage device type.
10–13	Retry count. Line 11 shows the maximum number of retries the device handler can perform. Line 12 tells the number of retries left before the transfer fails. If the number of remaining retries is 0, the transfer has failed. If the number of remaining retries is not 0, this usually indicates that a soft error has occurred, or that the transfer failed and the registers differed. In this example, with 6 retries remaining, the report was generated on the second retry. Line 13 tells how many times the error occurred with the same register contents.
15	Labels the section on device information. The lines that follow provide statistics on the device registers and address information.
16–19	Register contents. Each device has a number of hardware registers, the contents of which are listed in these lines.
21	I/O transfer type. Tells whether the I/O transfer was a read or write operation.
22	Device block number. Tells which device block the error occurred in.
23	Physical buffer start address. Tells the physical address in memory of the user data buffer for this I/O transfer.
24	Transfer size in bytes. Tells the size in bytes of the unit of data the device handler has attempted to transfer.

Memory Error-Log Report

There are two kinds of memory errors for which the Error Logger creates reports: memory parity errors and cache memory errors. An example of a memory-parity error report follows. As with the storage-device report, this listing is numbered here to aid in describing its contents. The actual listings do not have line numbers.

Example Memory-Parity Error-Log Report

```
1  ****************************************************************
2  MEMORY PARITY ERROR
3  LOGGED 8-OCT-90 16:13:22
4  ****************************************************************
5
6  SOFTWARE STATUS INFORMATION:
7          SYSTEM REGISTERS:
8          PC   001026
9          PSW  000000
10         OCCURRENCES OF THIS ERROR WITH IDENTICAL PC  3.
11
12 DEVICE INFORMATION
13         MEMORY REGISTERS:
14         ADDRESS      CONTENTS
15         172100       100001
16
17         MEMORY SYSTEM ERROR REGISTER:    100000
18         CACHE CONTROL REGISTER:          000000
19         HIT/MISS REGISTER:               027000
20
21         ERROR TYPE IS MEMORY
```

Analysis of the Example Memory-Parity Error-Log Report

The following is a line-by-line analysis of the preceding memory-parity report.

Line	Explanation
1–4	Report header. Tells the date and time the error was logged.
7–10	System register contents. Gives the contents of the program counter and the processor status word at the time of the error, as well as the number of times the program counter was the same for this error.
13–15	Memory parity register contents. Identifies your system's memory parity control and status register(s) and gives their contents.
17–19	Cache memory register contents. This information is displayed for both a memory parity error and a cache memory error if your system includes cache memory. See the *PDP–11 Processor Handbook* for more information on the cache memory registers.
21	Error type. Tells whether the error was a memory error or a cache memory error (see the following cache memory report for cache memory statistics).

Example Cache-Memory Error-Log Report

The preceding line-by-line analysis of the memory-parity report also applies to the cache-memory report. Line 21 indicates that the memory error was in cache memory.

```
1  ****************************************************************
2  CACHE MEMORY ERROR
3  LOGGED 8-OCT-90 16:21:20
4  ****************************************************************
5
6  SOFTWARE STATUS INFORMATION:
7          SYSTEM REGISTERS:
8          PC   001026
9          PSW  000000
10         OCCURRENCES OF THIS ERROR WITH IDENTICAL PC  3.
11
12 DEVICE INFORMATION
13         MEMORY REGISTERS:
14         ADDRESS      CONTENTS
15         172100       100001
16
17         MEMORY SYSTEM ERROR REGISTER:    000200
18         CACHE CONTROL REGISTER:          000000
19         HIT/MISS REGISTER:               000032
20
21         ERROR TYPE IS CACHE
```

Summary Error-Log Report

The summary error-log report provides statistics for all the devices the Error Logger supports. These statistics include counts for successful and unsuccessful I/O transfers for storage devices, and error counts for memory errors. The report consists of three sections:

- Device statistics (A)
- Memory statistics (B)
- Report-file environment and error count (C)

Three example error-log report summaries follow, one for each section of a summary report.

A: Example Summary for Device Statistics

```
1  ****************************************************************
2  DEVICE STATISTICS
3  LOGGED SINCE 8-OCT-90 16:01:12
4  ****************************************************************
5
6  UNIT IDENTIFICATION
7         PHYSICAL UNIT NUMBER                                 000000
8         TYPE                                      RL11/RL02/RL02
9
10 DEVICE STATISTICS FOR THIS UNIT:
11        NUMBER OF ERRORS LOGGED                 0.
12        NUMBER OF ERRORS RECEIVED               0.
13        NUMBER OF READ SUCCESSES               65.
14        NUMBER OF WRITE SUCCESSES               4.
15
16  UNIT IDENTIFICATION
17        PHYSICAL UNIT NUMBER                                 000000
18        TYPE                                            RX211/RX02
19
20  DEVICE STATISTICS FOR THIS UNIT:
21        NUMBER OF ERRORS LOGGED                 1.
22        NUMBER OF ERRORS RECEIVED               1.
23        NUMBER OF READ SUCCESSES                0.
24        NUMBER OF WRITE SUCCESSES               0.
25
26  UNIT IDENTIFICATION
27        PHYSICAL UNIT NUMBER                                 000001
28        TYPE                                            RX211/RX02
29
30  DEVICE STATISTICS FOR THIS UNIT:
31        NUMBER OF ERRORS LOGGED                 0.
32        NUMBER OF ERRORS RECEIVED               0.
33        NUMBER OF READ SUCCESSES                2.
34        NUMBER OF WRITE SUCCESSES               0.
```

The Error Logger provides summary statistics for each device. Notice that for each device, the count of the number of errors logged and the count of the number of

errors received can be different. Sometimes, the error logger may receive an error but be unable to log it. This is usually due to full buffers or some other momentary software limitation. However, even if the Error Logger is unable to log an error, it is able to inform you of this fact.

B: Example Summary for Memory Statistics

This section of the report immediately follows the section on device statistics.

```
1  ****************************************************************
2  MEMORY STATISTICS
3  LOGGED SINCE 8-OCT-90 16:01:12
4  ****************************************************************
5
6  STATISTICS:
7          NUMBER OF MEMORY PARITY ERRORS                    3.
8          NUMBER OF CACHE ERRORS                            0.
```

C: Example Summary for File Environment and Error Count

This section of the report immediately follows the section on memory statistics.

```
1  REPORT FILE ENVIRONMENT:
2          INPUT FILE                    DL0:ERRLOG.DAT
3          OUTPUT FILE                   LP :      .LST
4          OPTIONS                                 /A
5          DATE INITIALIZED              8-OCT-90
6          DATE OF LAST ENTRY            8-OCT-90
7
8  TOTAL ERRORS LOGGED                          15.
9  MISSED REPORTS (TASK NOT READY)              11.
10 MISSED REPORTS (BUFFER FULL)                  0.
11 MISSED REPORTS (FILE FULL)                    0.
12 UNKNOWN DEVICE STATISTICS ENTRIES             0.
13 UNKNOWN ERROR RECORD ENTRIES                  0.
```

The segment of the report-file environment shown above provides information concerning the input report-file name (usually ERRLOG.DAT or ERRLOG.TMP) and the output report-file name (any name that you specify in the initial ERROUT command line).

The following is an analysis of the preceding C summary report.

Line	Explanation
5	The date when the input report file was initialized.
6	The date of the last error entry to the input report file.
8-13	Additional error count statistics. Lines 9 through 11 count the number of missed reports. A missed report is an I/O transfer or error for which the Error Logger was unable to gather information because ERRLOG was running but ELINIT had not been run, the internal buffer was full, or the ERRLOG.DAT statistics file was full.

Summary Error-Log Report

Line	Explanation
12	Count of unknown device statistics entries. An unknown device statistics entry occurs when ERROUT does not recognize the device identifier byte the EL program recorded in the statistics portion of the ERRLOG.DAT file. (All Digital-distributed device handlers that support error logging can be identified by ERROUT. This problem occurs most often with user-written handlers. See the *RT–11 Volume and File Formats Manual* for details on adding a device to ERROUT.)
13	Count of the unknown error record entries. This condition occurs when the ERROUT task cannot identify a device error recorded in the ERRLOG.DAT file. (Again, this condition occurs most often with user-written handlers.)

(T)MSCP Error-Log Reports

Beginning with RT–11, V5.5, RT–11 has 12 different MSCP error-log reports, depending on which error packet DU passes to the Error Logger:

- Last Fail
- Status Address Register
- Controller Error Log
- Host Memory Access Error Log
- Disk Transfer Error Log
- Standard Disk Interface (SDI) Error Log
- Small Disk Error Log
- Bad Block Replacement Attempt Error Log
- Tape Errors (TMSCP only)
- Standard Tape Interface (STI) Communications or Command Failure (TMSCP only)
- STI Formatter Error Log (TMSCP only)
- STI Drive Error Log (TMSCP only)

The header information in a (T)MSCP error-log report is the same as that of any other storage-device error-log report. The rest of the report has a format different from other storage-device error-log reports. The following sections show an example and an analysis of an MSCP Standard Disk Interface packet error-log report.

Example of a DU MSCP Error-Log Report

The following is an example of a DU MSCP error-log report, illustrating the Standard Disk Interface packet:

```
ERROR LOG REPORT RT-11 V05.17  - COMPILED 6-JUL-90 02:34:54   REPORT 10.

********************************************************************
MSCP DEVICE ERROR
LOGGED 6-JUL-90 02:34:26
********************************************************************

Unit Identification
        RT-11 Unit Number                                       000000
        Device Type                                            DU/MSCP

Software Status Information
        Maximum Retries    0.
        Remaining Retries  0.
        Occurrences Of This Error With Identical Registers  1.

Device Information
        Active Function        NON-STANDARD TRANSFER
        Block                                  0.
        Physical Buffer Address             00000000
        Transfer Size In Bytes                   512.


CSR Address:                        172150

SAR Contents:                       000000 Controller On Line

(T)MSCP Packet Type                 SDI (Standard Disk Interface)
Error Packet

UQSSP Envelope                      000070 000007

MSCP Packet                         000000 000000 000002 000000
                                    040403 000353 045504 141722
                                    000201 000406 000004 000003
                                    007151 000000 000000 001005
                                    000406 000000 005424 000000
                                    000000 000000 002013 000200
                                    005000 147000 006004 017043
                                    172150 000000 000000 000000

Unit Hardware Version               000
Unit Number                         000002
Message Flags                       001    Operation Continuing
Status/Event Code                   000000 Success
Controller Model                    005    UDA50-A
Controller Software Version         000
Controller Hardware Version         000
Unit Identifier                     3689.
Unit Model                          005    RA 81
Unit Software Version               000
Unit Hardware Version               000
Pack/HDA Serial Number              0.
MSCP Logical Block Number           000000 000000
Generic Drive Status                002013 RUN/STOP Switch In
                                           Port Switch In
                                           Log Information in Extended Area
                                    000200 Drive Error
Extended Area                       000000
Extended Drive Status/Error Info    005000 147000 006004 017043
Drive Error Code                    23 (HEX)
Front Panel Fault Code              1E (HEX)
```

Analyzing the Example DU MSCP Report

The following is an item-by-item analysis of the preceding example of a DU MSCP report:

Item	Explanation
Report header	Includes the date and time the error was logged.
Unit identification	Identifies the drive number, the device handler, and the storage device type.
Software-status information	Identifies the retry count (for DU & MU is always 0)
Device information	Under very rare circumstances and only for the DU or MU report, the values under Device Information can be all zeros. If that happens, the (T)MSCP information is still valid and the values for Device Information should be ignored.
I/O transfer type	Tells whether the I/O transfer was a READ or a WRITE operation, or a NONSTANDARD TRANSFER (for an RT–11 special function of *type* equals 'other' or 'motion', or a special directory operation).
Device block number	Tells in which device block the error occurred. Note that if the I/O operations was an RT–11 special function or a special directory operation, the value displayed may not actually be an address. The error logger simply translates and displays whatever value is contained in the packet.
Physical buffer address	Tells the physical address in memory of the user data buffer for this I/O transfer. Note that if the I/O operations was an RT–11 special function or a special directory operation, the value displayed may not actually be an address. The error logger simply translates and displays whatever value is contained in the packet.
Transfer size in bytes	Tells the size in bytes of the unit of data the device handler has attempted to transfer. Note that if the I/O operations was an RT–11 special function or a special directory operation, the value displayed may not actually be an address. The error logger simply translates and displays whatever value is contained in the packet.
The following fields are of interest only to Digital Customer Services representatives.	
CSR address	The address of the Control Status Register at which your device handler is installed. This is the base address of the set of registers in the I/O page belonging to the device that caused the error.
Contents of status address (SA) register	

Analyzing the Example DU MSCP Report

Item	Explanation
	MSCP Description Section
	The description of the remainder of the MSCP or TMSCP packet that contains the error statistics is intended for use by only Digital maintenance personnel.
	The type of MSCP error packet.
	The 12 possible types of MSCP error packets are listed in the section *(T)MSCP Error-Log Reports*.
	UQSSP envelope.
	(T)MSCP packet.
	A listing of the contents of the packet. This contains various parameters required by the system to control the device, selected elements from the MSCP packet that help identify the error data.
	MSCP Drive Number.
	Identifies the MSCP unit number of the device.
	Note: The rest of the fields are related to the hardware that generated the error and are not described.

Chapter 10
File-Exchange Utility (FILEX)

The File-Exchange Utility (FILEX) is a general file-transfer program that converts files from one format to another so that you can use them with various operating systems.

Supported FILEX Devices

You can copy files between any block-replaceable, RT–11 directory-structured device and any device listed in Table 10–1.

Table 10–1: Supported FILEX Devices

Device	Valid as Input	Valid as Output
PDP–11 DOS/BATCH DECtape	X	X
DOS/BATCH disk	X	
RSTS DECtape	X	X
DECsystem–10 DECtape	X	
Interchange diskette (RX01, RX02 single-density, PDT–11/150)	X	X

NOTE

FILEX does not support magtapes, cassettes, or double-density diskettes in any operation.

You can transfer only one file at a time to interchange diskette format.

Defaults and Wildcards

The default device for all FILEX operations is DK. You can use wildcards when transferring from interchange to RT–11 format. However, you cannot use embedded wildcards in any file name or file type. For example, the following line specifies a valid file specification:

```
**.MAC
```

The next line is an invalid file specification for FILEX:

```
*T%ST.MAC
```

Operating Systems and File Formats

FILEX can transfer files created by four different operating systems:

- RT–11
- DECsystem–10
- Universal interchange format (IBM) system (see the *RT–11 Software Support Manual*)
- DOS/BATCH (PDP–11 Disk Operating System)

You can use the following three data formats in a transfer:

- ASCII
- Image
- Packed image

ASCII files conform to the American Standard Code for Information Interchange in which each character is represented by a 7-bit code. In ASCII mode, FILEX deletes null and rubout characters, as well as parity bits.

Because the file structure and data formats for each system vary, options are needed in the command line to indicate the file structures and the data formats involved in the transfer. See the section *FILEX Options* for descriptions of these options. FILEX assumes that all devices are RT–11-structured. You can use options to indicate otherwise.

Note that if you attempt to use RT–11 volumes for both input and output, FILEX generates an error message.

Calling and Terminating FILEX

To call FILEX from the system device, respond to the keyboard monitor dot prompt (.) by typing:

```
.R FILEX RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to enter a command.

Type CTRL/C to halt FILEX when it is waiting for console terminal input and return control to the monitor. To restart FILEX, type R FILEX or REENTER in response to the monitor's dot.

FILEX Option Types

FILEX has the following four categories of options:

- **Transfer**

  Transfer options direct FILEX to copy data in one of the following modes: ASCII, image, or packed-image.

- **Operation**

  Operation options transfer data, delete files, produce listings, and initialize device directories. FILEX accepts one transfer option and one operation option in a single command.

- **Modifier**

  Modifier options modify the file transfer. For example, when you use the /Y option to modify the /Z option, FILEX suppresses the */Init are you sure?* message.

- **Device**

  Device options indicate the formats of devices that are involved in a transfer. You can specify one device option for each file involved in the transfer. Device options must follow the device and file name to which they apply; other options can appear anywhere in the command line.

FILEX Option Summary

Table 10–2 lists the options that initiate various FILEX operations. See the following sections for more complete option descriptions.

Table 10–2: FILEX Options

Option	Type	Function
/A	Transfer	Indicates a character-by-character ASCII transfer in which FILEX deletes rubouts and nulls. If you use /U with /A, FILEX also ignores all sector boundaries on the diskette and assumes that records are to be terminated by a line feed, vertical tab, or form feed. If you use /A with /T, FILEX assumes that each PDP–10 36-bit word contains five 7-bit ASCII bytes. The transfer terminates when a CTRL/Z is encountered. (This feature is included for compatibility with RSTS.) FILEX does not transfer the CTRL/Z.
/D	Operation	Deletes the file you specify from the device directory. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
/F	Operation	Produces a brief listing of the device directory on the terminal. It lists only file names and file types. FILEX can only list directories of block-replaceable devices, and those directories only on the console terminal.
/I	Transfer	Performs an image-mode transfer. If the input is DOS/BATCH, RSTS, or RT–11, the transfer is word-for-word. If the input is from DECsystem–10, /I indicates that the file resembles a file created on DECsystem–10 by MACY11, MACX11, or LNKX11 with the /I option. In this case, each PDP–10 36-bit word will contain one PDP–11 8-bit byte in its low-order bits. If input or output is an interchange diskette, FILEX reads and writes four diskette sectors for each RT–11 block.
/L	Operation	Produces a complete listing of the device directory on the console terminal, including file names, block lengths, and creation dates.
/P	Transfer	Performs a packed image mode transfer. If the input is DOS/BATCH, RSTS, or RT–11, the transfer will be word-for-word. If the input is from DECsystem–10, /P indicates that the file resembles a file created on DECsystem–10 by MACY11, MACX11, or LNKX11 with the /P option. In this case, each PDP–10 36-bit word will contain four PDP–11 8-bit bytes aligned on bits 0, 8, 18, and 26. This is the default mode. If the output is interchange diskette, FILEX writes the data as EBCDIC.
/S	Device	Indicates that the device is a valid DOS/BATCH or RSTS block-replaceable device.

Table 10–2 (Cont.): FILEX Options

Option	Type	Function
/T	Device	Indicates that the device is a valid DECsystem–10 DECtape.
/U[:size.]	Device	Indicates that the device is an interchange diskette. The *size* specifies the length of each output record, in characters. *Size.* is a decimal integer in the range 1–128. The default value is 80; *size* is not valid with an input file specification, or with /A or /I.
/V[:ONL]	Modifier	/V is used with /Z and /U[:size] together to write a volume identification on an interchange diskette during initialization. A volume identification can be up to six characters long. Using /V:ONL with /Z and /U[:size] changes only the ID and does not initialize the interchange diskette. You can also use /V[:ONL] with /F or /L to list the volume identification of an interchange diskette as well as its directory.
/W	Modifier	Transfers files in a single- or small-disk system. FILEX initiates the transfer, but pauses and waits for you to mount the volumes involved in the transfer.
/Y	Modifier	Used with /Z to suppress the *dev:/Init are you sure?* message.
/Z	Operation	Initializes the directory of the device you specify. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.

Deleting Files (/D)

You can use FILEX to delete files from DOS/BATCH DECtapes, RSTS DECtapes, and interchange diskettes.

To delete files, type a command with the following syntax:

filespec/D/option

where:

filespec	specifies the device, file name, and file type of the file to be deleted.
/D	is the delete option.
/option	can be either /S, for DOS/BATCH and RSTS block-replaceable devices, or /U, for interchange diskettes. You can also include the /W modifier option, if necessary.

Examples

1. This command deletes all files with the file type PAL on DECtape unit 0:

```
*DT0:*.PAL/D/S
```

2. This command deletes the file TABLE.OBJ from the DECtape on unit 2:

```
*DT2:TABLE.OBJ/D/S
```

3. This command deletes all files with an RNO file type from the interchange diskette on unit 0:

```
*DX0:*.RN/D/U
```

Listing Directories (/L)

You can list at the terminal a directory of any of the block-replaceable devices used in a FILEX transfer. The command syntax is:

device:/L/option

where:

device specifies the block-replaceable device. These are the valid device types:

DOS/BATCH, RSTS DTn:, RKn:
DECsystem–10 DTn:
Interchange diskette DXn: DYn:

/L is the listing option. (You can substitute /F if you want a brief listing of file names only.)

/option is /S, /T, or /U, and the /W modifier option if necessary. These are the valid format and option combinations:

DOS/BATCH, RSTS /S
DECsystem–10 /T
Interchange diskette /U

Examples

1. This example shows the complete disk directory for UIC[1,7] of the device RK1. The letter C following the file size on a DOS/BATCH or RSTS directory listing indicates that the file is contiguous:

```
 *RK1:/L/S
 18-FEB-91
BADB      .SYS       1       18-FEB-91
MONLIB    .CIL     175C      18-FEB-91
DU11      .PAL      45       18-FEB-91
VERIFY    .LDA      67C      18-FEB-91
```

2. This example is a command that lists all files with the file type PAL that are stored on DECtape unit 1:

```
*DT1:*.PAL/L/S
```

3. This command produces a brief directory listing of the interchange diskette on unit 0, giving file names only:

```
*DX0:/U/F
```

4. This command lists all files on the DECsystem–10 formatted DECtape on unit 1, regardless of file name or file type; with the /F, a brief directory is requested in which only file names display:

```
*DT1:*.*/F/T
```

Transferring Between RT–11 and DOS/BATCH or RSTS (/S)

You can transfer files between block-replaceable devices used by RT–11 and the PDP–11 DOS/BATCH system. Input from DOS/BATCH may be either disk or DECtape. You can use both linked and contiguous files.

If the input device is a DOS/BATCH disk, you should specify a DOS/BATCH user identification code (UIC) in the form [nnn,nnn]. The initial default value is [1,1]. The UIC you supply will be the default for all future transfers. If you do not specify a UIC, FILEX will use the current default UIC. Note that the square brackets ([ ]) are part of the UIC; you must type them when you specify a UIC.

Output to DOS/BATCH is limited to DECtape only. You do not need a UIC in a command line where you are accessing only DECtape. Individual users do not own files on DECtape under DOS. However, no error occurs if you do use a UIC. DECtape used under the RSTS system is valid as both input and output, since its format is identical to DOS/BATCH DECtape. You may use any valid RT–11 file storage device for either input or output in the transfer. The RT–11 device DK is assumed if you do not indicate a device.

An RT–11 DECtape can hold more information than a DOS/BATCH or RSTS DECtape. When you copy files from a full RT–11 tape to a DOS DECtape, some information may not transfer. In this case, an error message displays and the transfer does not complete.

When a transfer from an RT–11 device to a DOS DECtape occurs, the block size of the file can increase. However, if the file is later transferred back to an RT–11 device, the block size does not decrease.

To Transfer to RT–11

To transfer a file from a DOS/BATCH block-replaceable device or RSTS DECtape to an RT–11 device, type a command with the following syntax:

output-filespec=input-filespec/S[/option]

where:

output-filespec	specifies any valid RT–11 device, file name, and file type (if the device is not file structured, you may omit the file name and file type).
input-filespec	specifies the DOS/BATCH or RSTS device, UIC, file name, and file type to be transferred. (See Table 7–1 for a list of valid devices.)
/S	is the option that designates a DOS/BATCH or RSTS block-replaceable device. (This option must be included in the command line.)
/option	is one of the three FILEX transfer options listed in Table 10–2, and the /W modifier option if necessary.

To Transfer from RT–11

To transfer files from an RT–11 storage device to a DOS/BATCH or RSTS DECtape, type a command with the following syntax:

DTn:output-filename/S[/option]=input-filespec

where:

DTn:output-filename	specifies the file name and file type of the file to be created, as well as the DOS/BATCH or RSTS DECtape on which to store the file.
input-filespec	specifies the device, file name, and file type of the RT–11 file to be transferred.
/S	is the option that designates a DOS/BATCH or RSTS DECtape. (This option must be included in the command line.)
/option	is one of the three transfer options from Table 10–2, and the /W modifier option if necessary.

Examples

1. This command instructs FILEX to transfer a file called SORT.ABC from the RT–11 default device DK to a DECtape in DOS/BATCH or RSTS format on unit DT2. The transfer is in image mode:

   ```
   *DT2:SORT.ABC/S=SORT.ABC/I
   ```

2. This command allows a file to be transferred from DOS/BATCH (or RSTS) DECtape to the printer under RT–11. The transfer is done in ASCII mode:

   ```
   *LP:=DT2:FIL.TYP/S/A
   ```

3. This command causes the file MACR1.MAC to be transferred from the DOS /BATCH disk on unit 1, stored under the UIC [1,2], to the RT–11 device DK. [1,2] becomes the default UIC for any further DOS/BATCH operations:

   ```
   *DK:*.*=RK1:[1,2]MACR1.MAC/S
   ```

Transferring to RT–11 from DECsystem–10 (/T)

Any valid RT–11 device is a valid output device, but only the DECsystem–10 DECtape is a valid input device. To transfer files from DECsystem–10 format to RT–11 format, use this command syntax:

output-filespec=input-filespec/T[/option]

where:

output-filespec	specifies any valid RT–11 device, file name, and file type. (If the device is not file-structured, you can omit the file name and file type.)
input-filespec	specifies the DECtape unit, file name, and file type of the DECsystem–10 file to be transferred.
/T	is the option that signifies a DECsystem–10 DECtape. (When you use /T, and especially when you also use /A, the system clock loses time. Examine the time, and reset it if necessary with the TIME command.)
/option	is one of the three transfer options from Table 10–2, and the /W modifier option if necessary.

You cannot convert RT–11 files to DECsystem–10 format directly. However, there is a two-step procedure for doing this. First, run RT–11 FILEX and convert the files to DOS-formatted DECtape. Then run DECsystem–10 FILEX to read the DOS DECtape.

Examples

1. This command converts the ASCII file STAND.LIS from DECsystem–10 ASCII format to RT–11 ASCII format and stores the file under RT–11 on DECtape unit 2 as STAND.LIS:

   ```
   *DT2:STAND.LIS=DT1:STAND.LIS/T/A
   ```

   Transfers from DECsystem–10 DECtape to RT–11 may cause an <UNUSED> block to appear after the file on the RT–11 device. This is a result of the way RT–11 handles the increased amount of information on a DECsystem–10 DECtape.

2. This command indicates that all files on the DECsystem–10 formatted DECtape on unit 0 with the file type .LIS are to be transferred to the RT–11 system device using the same file name and a file type of .NEW. The /P option is the assumed transfer mode:

   ```
   *SY:*.NEW=DT0:*.LIS/T
   ```

   Files may not be transferred to RT–11 devices from a DECsystem–10 DECtape if a foreground job is running. This restriction is due to the fact that when FILEX reads DECsystem–10 files, it accesses the DECtape control registers directly instead of using the RT–11 DECtape handler.

Transferring Between RT–11 and Interchange Diskette (/U)

You can transfer files between block-replaceable devices used by RT–11 and interchange format diskettes. Files are transferred in one of three formats: ASCII, image, and packed image EBCDIC mode.

A universal diskette consists of 77 tracks (some of which are reserved), each containing 26 sectors numbered from 1 to 26. A sector contains one record of 128 or fewer characters. When an interchange diskette is in packed image mode, records always begin on a sector boundary. There is only one record per sector. If a record does not fill a sector, the remainder is filled with blanks. Since packed image EBCDIC mode is inefficient and wastes space, packed image mode is recommended only to read or write diskettes that must be compatible with IBM 3741 format. Packed image (EBCDIC) mode is generally compatible with IBM 3741 format. (Although IBM 3741 format supports error mapping of bad sectors and multivolume files, FILEX does not.) Packed image (EBCDIC) is the default mode, so you must use one of the options from Table 10–2 to specify ASCII or image mode. All records of a file must be the same size. You indicate this with the /U:size. option.

NOTE

File types are not usually recognized in interchange format; instead, a single, 8-character file name is used. However, to provide uniformity throughout RT–11, FILEX has been designed to accept a 6-character file name with a 2-character file type. If you transfer a file from RT–11 to interchange diskette, any 3-character file type is truncated to two characters.

To Transfer from RT–11

To transfer files from RT–11 format to interchange format, type a command with the following syntax:

output-filespec/U[:size.][/option]=input-filespec

where:

output-filespec	specifies the device, file name, and file type of the interchange file to be created. Note that you cannot use wildcards in the output file specification.
/U[:size.]	is the option that designates an interchange diskette. This option must be included in the command line. *Size.* specifies the length of each output record, in decimal integer in the range 1 to 128 (default is 80). *Size* is invalid with either /A or /I.
/option	is one of the three transfer options from Table 10–2, and the /W modifier option if necessary.

input-filespec specifies the device, file name, and file type of the RT–11 file to be transferred. The file name is six characters long, with a 2-character file type. Any 3-character file type is truncated to two characters.

To Transfer to RT–11

To transfer files from interchange diskette to RT–11 format, type a command with the following syntax:

output-filespec=input-filespec/U[/option]

where:

output-filespec specifies the device, file name, and file type of the RT–11 file to be created. Note that you can use wildcards as input.

input-filespec specifies the device, file name, and file type of the interchange file to be transferred.

/U is the option that designates an interchange diskette. (This option must be included in the command line.)

/option is one of the three transfer options from Table 10–2, and the /W modifier option if necessary.

Examples

1. This command transfers the file IVAN.CAT from RT–11 RK05 unit 2 to the diskette on unit 1. The transfer is done in exact image mode (indicated by /I), ignoring all sector boundaries:

   ```
   *DX1:IVAN.CA/U/I=RK2:IVAN.CAT
   ```

2. This command instructs FILEX to transfer the file BENMAR.FRM from the RT–11 disk unit 2 to the diskette on unit 0, and rename it KENJOS.JO. The /U option indicates that the format is to be changed from ASCII to the interchange format. There will be one record per sector of 128 or fewer characters. If there are fewer than 128 characters, the remainder of the sector will be filled with spaces:

   ```
   *DX0:KENJOS.JO/U=RK2:BENMAR.FRM
   ```

3. This command transfers the file TYPE.SET from RT–11 diskette unit 0 to the interchange diskette on unit 2. The exchange converts ASCII to interchange format, putting a maximum of seven (indicated by :7.) characters into each sector until the entire record has been transferred. Records in excess of seven characters will be broken up and placed in succeeding sectors on the diskette. New records always begin on a sector boundary; returns and line feeds are discarded. However, if you use /A or /I, FILEX ignores boundary limits and preserves returns and line feeds:

   ```
   *DX2:TYPE.SE/U:7.=DX0:TYPE.SET
   ```

   File TYPE.SET before transfer:

   ```
   ABCDEFGHIJKLMN
   ```

File TYPE.SET after transfer:

```
ABCDEFG----(spaces up to 128 characters) Sector 1
HIJKLMN----(spaces up to 128 characters) Sector 2
```

4. This command copies file IVAN.CA from the interchange diskette on unit 1 to the RT–11 printer, treating the input as ASCII characters. Note that once a record has been divided into sectors, it cannot be transferred back to its original size:

```
*LP:=DX1:IVAN.CA/U/A
```

Writing an Interchange Volume ID (/V[:ONL])

The /V option enables you to write a volume identification on an interchange diskette when it is initialized. This option is used with the /U[:size] and /Z options together. You can also use /V[:ONL] with /L or /F to list a volume ID.

When you use this option, FILEX prompts you for a volume ID. Respond by typing a volume identification of up to six characters. Any string over six characters is truncated. If you type only a return in response to the volume ID prompt, the default volume ID RT11A is written on the interchange diskette.

Use /V:ONL to change only the volume ID without initializing the interchange diskette.

Examples

1. This command initializes an interchange diskette and writes a volume identification:

```
*DX0:/Z/U/V
 Volume ID? Nancy
```

2. This command changes only the volume ID of an interchange diskette:

```
*DX0:/Z/U/V:ONL
 Volume ID change; are you sure? Y
 Volume ID? Nancy
```

Starting and Then Pausing a Transfer (/W)

The /W option permits you to replace the system volume with another volume during an operation. You can use the /W option for a delete, directory listing, and initialization operation on a single-disk system, or to copy files between volumes when the system volume is neither the input nor the output volume if you have two drives available. When you use the /W option, you cannot use wildcards in the input specification.

When you use the /W option, FILEX guides you through a series of steps in the process of completing the operation. After you enter the initial command string, FILEX displays a message telling you which volume to mount. After you complete each step, type Y or any string beginning with Y followed by a RETURN to proceed to the next step. If you type N or any string beginning with N, or CTRL/C, FILEX prompts you to mount the system volume if you have removed it and the operation is not performed. Any other response causes the message to repeat.

When the operation is complete, FILEX displays a message instructing you to mount your system volume. Mount the system volume and type Y or any string beginning with Y followed by a RETURN. If you type any other response, FILEX prompts you to mount the system volume until you type Y.

When you use /W, make sure that FILEX is on your system volume.

The Procedure for Copying Files with /W

1. With your system volume mounted, enter a command string according to the FILEX syntax. After you have entered the command string, FILEX responds with the message:

   ```
   Mount input volume in <device>; Continue?
   ```

2. Type Y or any string beginning with Y followed by a RETURN to continue the operation when you have mounted the input volume. FILEX then displays:

   ```
   Mount output volume in <device>; Continue?
   ```

3. Type Y or any string beginning with Y followed by a RETURN to continue the operation after you have mounted the output volume.

4. When the file transfer is complete, FILEX displays the following message if you had to remove the system volume from <device>:

   ```
   Mount system volume in <device>; Continue?
   ```

5. Type Y or any string beginning with Y followed by a RETURN to terminate the copy operation. If you type any other response, FILEX prompts you to mount the system volume until you type Y.

Initializing Directories (/Z)

You can also use FILEX to initialize the directories of DOS/BATCH DECtapes, RSTS DECtapes, and interchange diskettes.

Use this command syntax:

device:/Z/option[/Y]

where:

device	specifies the DOS/BATCH or RSTS DECtape, or the interchange diskette to be zeroed.
/Z	is the initialize option.
/option	can be either /S, for DOS/BATCH and RSTS DECtapes, or /U, for interchange diskettes. You can also include the /W modifier option, if necessary.
/Y	inhibits the FILEX confirmation message.

Examples

1. This command directs FILEX to initialize the directory of the interchange diskette on unit 0:

   ```
   *DX0:/Z/U
   ```

   FILEX displays a confirmation message:

   ```
   DX0:/Initialize; are you sure?
   ```

   Respond with a Y or any string beginning with Y followed by a RETURN for initialization to begin. Any other response aborts the command.

2. This command initializes the DECtape on unit 1 in DOS/BATCH (RSTS) format. Note that by using the /Y option you suppress the confirmation message:

   ```
   *DT1:/Z/S/Y
   ```

NOTE

The directory of an initialized interchange diskette has a single file entry, DATA, that reserves the entire diskette. You must delete this file before you can write any new files on the diskette. This is necessary for IBM compatibility. Do this by using the following command:

```
*DX0:DATA/D/U
```

DCL Equivalents of FILEX Utility Operations

Table 10–3 lists the DCL COPY, DELETE, DIRECTORY, and INITIALIZE commands that are equivalent to FILEX utility operations.

Table 10–3: DCL Equivalents of FILEX Utility Operations

CSI Option	DCL Command	DCL Option
/A	COPY	/ASCII
/D	DELETE	
/F	DIRECTORY	/FAST
/I	COPY	/IMAGE
/L	DIRECTORY	
/P	COPY	PACKED
/S	COPY	/DOS
/T	COPY	/TOPS
/U[:size]	COPY	/INTERCHANGE[:size]
/V[:ONL]	DIRECTORY INITIALIZE	/VOLUMEID[:ONLY]
/W	COPY DELETE DIRECTORY INITIALIZE	/WAIT
/Y	INITIALIZE	/NOQUERY
/Z	INITIALIZE	

Chapter 11
Volume Formatting Utility (FORMAT)

Formatting Volumes

The Volume Formatting Utility (FORMAT) formats a volume to make it usable for RT–11. FORMAT does this by writing on each block in that volume a header block containing data the device controller uses to transfer information to and from that block.

Formatting Disks and Diskettes

FORMAT supports the following devices:

- RX02 and RX33 diskettes
- RK05 disks, RK06/RK07 disks
- RD50/RD51/RD52/RD53 disks (DW handler)
- RD50, RD31, RD51, RD52, and RD53 (These disks are supported for CTI Bus-based processors, by commands FORMAT, FORMAT/VERIFY, and FORMAT /VERIFY:ONLY.)
- All of the preceding plus RX50, RL01, and RL02 for /VERIFY:ONLY

When you convert a single-density diskette to double density, or the reverse, FORMAT writes media density marks on each block of the diskette. You can format a diskette only in a double-density diskette drive, DY. If you attempt to format a diskette in a single-density diskette drive, DX, FORMAT displays an error message.

Reformatting with the FORMAT program can also eliminate bad blocks that disks and diskettes sometimes develop as a result of age and use. Although formatting does not ensure that each bad block will be eliminated, formatting reduces the number of bad blocks.

NOTE

Be aware that FORMAT will destroy any data that currently exists on the disk.

Do not format or verify a volume while a foreground job is loaded. To do so will cause data on the volume to be written over and corrupted, and will crash either the foreground job or the system.

Formatting Extended Device Units

To accommodate FORMAT support for extended device units greater than 7, FORMAT recognizes the command-line syntax:

FORMAT device-unit-number

For example:

```
FORMAT D10.
```

Formatting Devices at Nonstandard Addresses

Formatting devices at nonstandard addresses will occur automatically, based on the CSR location specified in the device handler. For DU devices, any supported number of controllers is allowed. For all other devices, only a single controller is allowed.

Although hardware is directly accessed for some devices, FORMAT uses the handler file contents to determine CSR address.

Initializing after Formatting

After you have formatted a disk, issue the INITIALIZE command to prepare the volume for use with RT–11. See the *RT–11 Commands Manual* for more information on the INITIALIZE command.

Calling and Terminating FORMAT

To call FORMAT from the system device, issue the following command line:

```
.R FORMAT RET
```

The Command String Interpreter (CSI) displays an asterisk (*) at the left side of the terminal screen and prompts for a command line. When you press RETURN in response to the prompt, FORMAT displays its current version number. Press CTRL/C to halt FORMAT and return control to the monitor when FORMAT is waiting for input from the terminal.

You cannot halt FORMAT during an operation by pressing CTRL/C two times. If you use some other means to interrupt the program during a formatting operation, the disk or diskette involved will not be completely formatted. You will have to restart the operation on the same disk or diskette and allow it to run to completion.

FORMAT Command-Line Syntax

FORMAT accepts one device specification (physical or logical device name) and one or more options. You can format an RK05 disk located in any unit (0–7) of device RK. A diskette must be mounted on an RX02 device (device DY), but it can be located in any unit (0–3) of that device. You cannot format diskettes on an RX01 device.

See Chapter 1 for more information on the general command-line syntax acceptable to system utility programs.

Default Format

To format diskettes in double-density mode, specify the device name in the command line. You can also use /Y to suppress the query message, and /W to pause for a volume substitution. The following example formats the diskette in DY device unit 1 as a double-density diskette:

```
DY1:
DY1:/FORMAT-Are you sure? Y RET
?FORMAT-I-formatting complete
```

To format an RK05 or RK06/07 disk, specify the device name in the command line. You can also use /Y to suppress the query message and /W to pause for a volume substitution. If necessary, you can abort the /W (WAIT) at any time. The following example formats an RK05 disk in RK device unit 1:

```
RK1:
RK1:/FORMAT-Are you sure? Y RET
?FORMAT-I-formatting complete
```

When you format an RK06 or RK07 disk, FORMAT lists the block numbers of all the bad blocks in the manufacturer's bad block table and in the software bad block table.

FORMAT Confirmation Prompts and Messages

FORMAT automatically displays the message *<device>:/Are you sure?* before it begins any operation. The <device> that displays the message is the physical-device name you specify in the command line. However, if you use a logical-device name in the command line, the device name that FORMAT displays in the confirmation message will be different from the name you enter. If you want the operation to continue, type Y or any string beginning with Y, then press RETURN in response to the confirmation message. Type N or press CTRL/C to discontinue the formatting operation. Any other response causes FORMAT to repeat the prompt.

The error message, *?FORMAT-U-Channel in use,* indicates an internal FORMAT error. If you receive that message, reboot your system and try the operation again. If the error reoccurs, get a new copy of FORMAT.SAV and retry the operation. If the error persists, submit a Software Performance Report to Digital.

Attempting to format a disk that is not mounted returns the error message, *?FORMAT-F-Device not ready.* If you receive that message and your disk is not mounted, mount your disk and be sure it is up to speed.

You can use FORMAT from a command file. To satisfy the message *<device>:/Are you sure?*, type Y as the next line of the command file immediately following the FORMAT command line. You can completely suppress the confirmation message by using the /Y option in the FORMAT command line. If you use /Y, you do not need to enter the Y on the next line.

If you try to format a volume that contains protected files, the system displays the following message:

```
Volume contains protected files; Are you sure?
```

Type Y or any string beginning with Y to continue the formatting operation. Type N or any string beginning with N, or press CTRL/C to abort the operation. Any other response causes the message to repeat.

FORMAT Option Descriptions

Pattern Verification Option (/P:value)

When you use the /P:value option with /V[:ONL] in the command line, you can specify the 16-bit word pattern you want FORMAT to use when it performs volume verification. The *value* argument is an octal integer in the range 0 to 177777 that specifies the pattern or successive patterns you want FORMAT to use.

Verification Bit Patterns

The following table lists the FORMAT verification patterns with the corresponding *values*.

Pattern	Bit Set	Value	16-Bit Pattern
1	0	1	000000
2	1	2	177777
3	2	4	163126
4	3	10	125252
5	4	20	052525
6	5	40	007417
7	6	100	021042
8	7	200	104210
9	8	400	155555
10	9	1000	145454
11	10	2000	146314
12	11	4000	162745
13	12	10000	†
14	13	20000	†
15	14	40000	†
16	15	100000	†

†These patterns are reserved for future use. Currently, these bit patterns run the default bit pattern (pattern 8).

The number you specify for the *value* of /P:value indicates the value for the bit patterns to be run during the verification. The bits set by the /P:value option select the patterns to be run. The preceding table shows which bit, when set, corresponds to each 16-bit verification.

To calculate an equivalent number for *value*, convert the bit set to an octal number. For example, FORMAT runs pattern 3 when bit 2 is set. When bit 2 alone is set, the equivalent octal number is 4.

To run more than one bit pattern, add the *value*s for the patterns you select. For example, suppose you want to run bit patterns 1, 3, and 5. The corresponding values are 1, 4, and 20, for a sum of 25. This is the *value* you would specify with /P to run all three bit patterns.

FORMAT converts the number you specify into a binary number; the number of each set bit specifies which patterns to run. The number 25 translates to the binary number 010 101. In the number 010 101, bits 0, 2, and 4 are set. If you specify /P:777, FORMAT runs patterns 1 through 9 during verification. If you do not use the /P:value option, FORMAT runs only pattern 8.

FORMAT runs each pattern successively. After it completes verification, FORMAT displays each bad block found during each verification pass. The format of the verification report is:

```
PATTERN #x
----------
nnnnnn
```

In the preceding example, *x* specifies the pattern number, and *nnnnnn* is the bad block number. FORMAT makes a separate verification pass for each pattern it runs, and reports on each pass.

In the following example, the command line formats volume RK1 and verifies it with patterns 4, 5, and 6:

```
*RK1:/V/P:70 RET
RK1:/FORMAT-Are you sure? Y RET
?FORMAT-I-formatting complete
PATTERN #6
PATTERN #5
PATTERN #4
?FORMAT-I-Verification complete
*
```

The command line in the next example verifies volume DL0 with pattern 2:

```
*DL0:/V:ONL/P:2
DL0:/VERIFY-Are you sure? Y RET
PATTERN #2
?FORMAT-I-Verification complete
```

Single-Density Option (/S)

Use /S to format a diskette in single-density mode. You can also use /Y to suppress the query message and /W to pause for a volume substitution.

The following example formats the diskette in DY device unit 1 as a single-density diskette:

```
*DY1:/S
DY1:/FORMAT-Are you sure? Y RET
?FORMAT-I-formatting complete
*
```

Verification Option (/V[:ONL])

Use the /V[:ONL] option to provide a verification of all blocks on a volume immediately following formatting. If you use the optional argument *:ONL*, FORMAT executes only the verification procedure.

Although FORMAT can format only a limited assortment of storage volumes, the /VERIFY:ONLY option can perform the write/read verify operation on the following devices: DL, DM, DU, DX, DW, DY, DZ, and RK.

When verifying a storage volume, FORMAT first writes a 16-bit word pattern on each block of the specified volume, and then reads each pattern. For each read or write error it encounters, FORMAT displays the block number for each block that generated the error.

NOTE

FORMAT will destroy data on any storage volume it verifies.

The following command line uses /V to verify an RK05 disk after formatting:

```
*RK0:/V RET
RX0:/FORMAT-Are you sure? Y RET
?FORMAT-I-formatting complete
PATTERN #8
?FORMAT-I-Verification complete
```

The next example uses /V:ONL to verify, but not format, an RX02:

```
*DX1:/V:ONL RET
DX1:/VERIFY-Are you sure? Y RET
PATTERN #8
?FORMAT-I-Verification complete
```

Wait Option (/W)

Before formatting begins, use /W to pause while you substitute a second volume for the disk specified in the command line, a useful technique for single-disk systems.

After the FORMAT program has accepted your command line, it pauses while you exchange volumes. Type Y or any string beginning with Y, then press RETURN to the *Continue?* prompt when you are ready to begin formatting. If you type N or any string beginning with N, or press CTRL/C, the operation is discontinued and the monitor dot prompt (.) displays. Any other response causes the message to repeat.

When formatting completes, the program pauses again while you replace the original volume. Respond to the *Continue?* prompt by typing Y or any string beginning with Y, then press RETURN.

You can combine /W with any other option. The following example formats the diskette in DY: device unit 1 as a single-density diskette:

```
*DY1:/W/S
DY1:/FORMAT-Are you sure? Y RET
Mount input volume in <dev:>;  Continue? Y RET
?FORMAT-I-formatting complete
Mount system volume in <dev:>;  Continue? Y RET
*
```

When you use the /W option, make sure that FORMAT is on the system volume.

No Query Option (/Y)

Use /Y to suppress the *Are you sure?* confirmation message FORMAT displays before each operation begins. When you use /Y, formatting begins as soon as FORMAT accepts and interprets your command line.

The following example formats the diskette in DY: device unit 1 as a double-density diskette:

```
*DY1:/Y
?FORMAT-I-formatting complete
*
```

FORMAT Option Summary

Table 11–1 summarizes the options you can use with the FORMAT program. You can combine these options in any order.

Table 11–1: FORMAT Option Summary

Option	Function
None	If you do not supply an option, FORMAT formats the volume you specify. If you specify an RX02 diskette, the default operation that occurs is double-density diskette formatting. You can use /Y and /W with the default operation.
/P:value	Pattern verification option, where *value* is an octal integer in the range 0 to 177777. The option specifies the 16-bit word pattern that FORMAT uses to write to the volume, and read from the volume, during the process of verification. If you do not use this option, FORMAT defaults to /P:200.
/S	Single-density option. This option formats a diskette in a single-density format.
/V[:ONL]	Verification option. When you specify /V in the command line, FORMAT first formats the specified volume, then verifies it. If you specify /V:ONL, FORMAT only verifies the specified device. You can use /V:ONL with RX02 diskettes, RL01/RL02 disks, TU56 (DECtape I), TU58 (DECtape II), and RD50/RD51 /RD52/RD53/RD54 hard disks.

Table 11–1 (Cont.): FORMAT Option Summary

Option	Function
/W	Wait option. Use this option to substitute another volume for the volume you specify in the command line, format the second volume, then replace the original volume. This option is invalid for RC25 disks, RD50/RD51/RD52 /RD53 disks, and RX50 diskettes.
/Y	No query option. This option suppresses the message, *Are you sure?*, that FORMAT automatically displays before each operation.

DCL Equivalents of FORMAT Operations

Table 11–2 lists the DCL commands that are equivalent to FORMAT utility operations.

Table 11–2: DCL Equivalents of FORMAT Utility Operations

CSI Option	DCL Command(s)	DCL Option(s)
/P:value	FORMAT	/PATTERN:value
/S	FORMAT	/SINGLEDENSITY
/V[:ONL]	FORMAT	/VERIFY[:ONLY]
/W	FORMAT	/WAIT
/Y	FORMAT	/NOQUERY

Chapter 12
Logical-Disk Subsetting Utility (LD)

The Logical-Disk Subsetting Utility (LD) lets you define and access logical disks, which is a way of subsetting physical disks. You define a logical disk by associating a logical-disk unit number with a file. Once defined, you can use DCL commands and utility programs to initialize, copy, and utilize these logical disks as if they were physical disks. For example, you can use the COPY/DEVICE command or the BACKUP/SUBSET command to copy logical disks as well as physical disks.

A Utility and a Device Handler

The LD utility is also a device handler. LD functions as a device handler when you load it and as a utility when you run it.

Uses for Logical-Disk Subsetting

- It is particularly useful when you work with large disks. Large disks can run out of directory-entry space before the volume is full. Since each logical disk has its own directory, dividing a physical disk into several logical disks creates more directory-entry space.
- It provides a convenient way to group files into logical collections.
- It allows you to perform some device and file operations more quickly.

See the *Introduction to RT–11* for tutorial information on using LD.

Calling and Terminating LD

To call LD from the system device, first be sure that LD is installed (see the INSTALL command in the *RT–11 Commands Manual*. Then, when running under an unmapped monitor, issue the command:

```
.R LD.SYS RET
```

When running under a mapped monitor, issue the command line:

```
.R LDX.SYS RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to issue a command line. When you press RETURN, LD displays its current version number and prompts you again for a command line. Press CTRL/C to terminate LD and return control to the monitor when LD is waiting for input. Press CTRL/C twice to terminate LD at any other time.

Command-Line Syntax

Specify the LD command line in the following general format:

input-specs/options

where:

input-specs specifies the files to be assigned as logical-disk units. You can specify up to six input file specifications in a command line. The default file type is DSK.

/options specifies an option from Table 12–1. You must specify at least one option in a command line, and you can specify more than one, as long as the operations you specify do not conflict.

LD Option Summary and DCL Equivalents

Table 12–1 summarizes the options you can use with logical disks.

Table 12–1: LD Option Summary

Option	Function
/A:name	Assigns a logical-device name to a logical disk. Must be used with /L.
/C	Verifies all logical-disk assignments against the files on the volumes currently mounted.
/L:value	Mounts a logical disk and associates it with a file on a disk, or dismounts a logical disk and disassociates it from a file on a disk.
/R:value	Write locks a logical disk. When you use /R:value, the logical disk you specify has read-only access.
/W:value	Write enables a logical disk. When you use /W:value, read/write access is allowed for the logical disk you specify.

Table 12–2 lists the DCL commands that are equivalent to LD utility operations.

Table 12–2: DCL Equivalents of the LD Utility Operations

CSI Option	DCL Command(s)	DCL Option(s)
/A:name	ASSIGN	–
/C	SET LDx	CLEAN
/L:value	MOUNT,DISMOUNT	–
/R:value	MOUNT,DISMOUNT	NOWRITE
/W:value	MOUNT,DISMOUNT	WRITE

LD Option Descriptions

Assign Logical-Device Name Option (/A:name)

Use the /A:name option with /L to assign a logical-device name to a logical disk. The *name* argument specifies the logical-device name, from one to three characters long, that you want to assign. The first character must be a letter. Optionally, you can include a colon after the logical-device name.

After you have assigned a logical-device name to a logical disk, you can refer to the logical disk by using the form LDn: or by using the logical-device name.

When the /A option is used in a command line, LD exits and does not prompt for another command line. Only one /L option is allowed in a command line containing the /A option.

The following command line assigns the logical-device name VOL to logical-disk unit 2 (LD2) when it is assigned to the file DK:LOGFIL.DSK:

```
LOGFIL.DSK/L:2/A:VOL
```

Validate Logical-Disk Assignments Option (/C)

The /C option validates all logical-disk assignments. When you use /C, LD checks the current logical-disk assignments against the files on volumes that are mounted.

The /C option is most useful after you have moved or removed files on a volume, or after you have removed a volume from a device. If a logical-disk file has moved, LD takes note of the new location so that you can continue to use that logical disk. If you have deleted a logical-disk file or the volume containing a logical-disk file is no longer mounted, LD disconnects the logical-disk assignment. In the case of a volume that you have removed, disconnection is temporary. You can reestablish the assignment when you remount the volume by using the /C option.

Note that after a squeeze (DUP /S option) or bootstrap operation, the system automatically performs a /C operation to update logical-disk assignments.

The /C option must be used alone on a command line. The following command line verifies current logical-disk assignments:

```
*/C
```

LD Option Descriptions

Define Logical-Disk Option (/L:value)

The /L:value option mounts a logical disk by associating it with a file on a device, or frees a logical-disk number so it can be associated with another file.

Use the following command syntax to mount a logical-disk unit number:

filespec/L:value

where:

filespec specifies the file to be associated with a logical-disk unit number. The file can reside on either a physical disk or another logical disk.

value specifies the logical-disk unit number to associate with the file. After it is mounted, the logical disk is referenced by using the device name LDvalue:. The *value* argument must be an integer in the range 0 through 7, unless you have SYSGENed your system for extended-unit support. With extended-unit support the range of *value* is 0 through 32_8.

NOTE

You must be careful to avoid accidentally destroying files while performing logical-disk subsetting. LD allows you to assign logical-disk unit numbers to both protected and SYS files, and to write to those files.

To free a logical-disk unit number from a file association, issue a command line having the following syntax:

/L:value

You can mount and dismount several logical disks on the same command line. For example, the following command associates logical-disk unit 0 with file MYFILE.DSK on DL1, logical-disk unit 4 with DATFIL.DSK on DU0, and dismounts logical-disk unit 2:

```
DL1:MYFILE/L:0,DU0:DATFIL.DSK/L:4,/L:2
```

You can also reassign a logical-disk unit number by simply specifying the /L option with the same logical-disk unit number and a different file name.

Write-Lock Logical-Disk Option (/R:value)

Use the /R:value option to write lock a logical disk. You then have read-only access to that logical disk. The *value* argument specifies the logical-disk unit number. This number must be an integer in the range 0 through 7, unless you have SYSGENed your system for extended-unit support. With extended-unit support the range of *value* is 0 through 32_8.

The default mode is /W (write-enabled).

The following command mounts logical disk unit 3 to JMS.TXT on DU1: and write locks it:

```
JMS.TXT/L:3/R:3
```

The next command write locks logical-disk unit 4:

```
/R:4
```

Write-Enable Logical-Disk Option (/W:value)

Use the /W:value option to write enable a logical disk. You then have read/write access to that logical disk. The *value* argument specifies the logical-disk unit number. This number must be an integer in the range 0 through 7, unless you have SYSGENed your system for extended-unit support. With extended-unit support the range of *value* is 0 through 32_8. /W (write-enabled) is the default mode.

The following command mounts logical-disk unit 5 to file JMS.DSK on DL0: and write-enables the new logical disk:

```
DL0:JMS/L:5/W:5
```

Chapter 13

The LET Substitution Utility (LET)

The LET Substitution Utility (LET) is an unsupported utility that enables you to substitute symbols for characters and strings in a DCL command line. The LET utility works with the SL (single-line) command-line editor to enable substitution. This provides a faster method of terminal input.

Enabling the LET Utility

To enable LET, type the following command:

```
.SET SL LET,KMON RET
```

This command assumes that SL is neither loaded nor on.

Defining Symbols for Substitution

To define a substitution, issue a LET command that equates a symbol with a character string. Use the following format:

LET _x=string

where

_ tells SL that you are defining a symbol. This prevents SL from trying to substitute a string for the character that follows.

x is a 1-character symbol that you want to equate with a string.

string is a character string.

For example, the following line equates the symbol # with the string DX:MYPROG.MAC:

```
.LET _#=DX:MYPROG.MAC RET
```

Having defined the preceding # symbol, whenever you type # on a command line, SL replaces it with DX:MYPROG.MAC. For example, type:

```
.MACRO # RET
```

The editor displays:

```
.MACRO DX:MYPROG.MAC
```

You can have up to five symbols concurrently defined, and each character string can include up to 32 (decimal) characters.

NOTE

Since LET substitutes your definitions for the symbols you define, you can no longer enter those symbols as themselves in a command line (since LET substitutes your definition when you use the symbol).

LET symbols remain defined even after you turn off your computer. This means, when you again use your computer (even if LET is not loaded), the symbols you defined at a previous time are still in the computer's memory. See the following section for how to delete the LET substitution.

Deleting LET Substitutions

To be able to use a LET defined symbol as itself in a command line (rather than as a substituted character or string), you must:

1. Use the /DELETE option to delete the assignment for X.
2. Unload LET.
3. Reload LET so that the deleted definition takes effect.

Defining Function Keys for Substitution

On keyboards that contain function keys (F1 through F20), LET supports defining keys F6 through F10, F14, and F17 through F20, as symbols for string substitutions. For example, the following LET substitution defines the F7 function key as the string *MACRO/LIST/CROSSREFERENCE*:

```
.LET F7=MACRO/LIST/CROSSREFERENCE RET
```

LET Options

The following table summarizes the LET options.

Command /Option	Function
LET /HELP	Displays help summary
LET /LIST	Displays current character assignments
LET x/DELETE	Deletes the assignment for x
LET /DELETE	Deletes all assignments with the query *Are you sure?*
LET /DEL:ALL	Deletes all assignments without requesting confirmation

Using LET in Your STRTxx.COM File

You can use the LET command in your STRTxx.COM file to delete all currently assigned characters and define those you will need for the work you are doing.

For example, you might include the following commands in your STRTxx.COM file:

```
.LET /DELETE:ALL
.LET _#=LET.MAC
.LET _$=LET
.LET _;=:
.LET _\=
```

This sequence would assign LET.MAC to # and LET to $. It would also cause the SL command-line editor to translate ; to : and \ to nothing.

Chapter 14
The Librarian Utility (LIBR)

The Librarian Utility (LIBR) lets you:

- Create, update, modify, list, and maintain *object* library files.

  The linker uses object libraries, as specified by the user, to resolve undefined external symbols.

- Create *MACRO* library files to use with the V03 and later versions of the MACRO–11 assembler.

  The assembler uses MACRO libraries as specified by the user, to resolve macro calls.

Library Files

A library file is a direct-access file (a file that has a directory) that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO–11 assembler can access them rapidly.

Each library contains:

- Library header
- Library directory (or global symbol table, or macro name table)
- One or more object modules or macro definitions.

  The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience.

The contents of the library file are determined by your needs. An example of a typical object library file is the default system library that the linker uses, SYSLIB.OBJ. An example of a macro library file is SYSMAC.SML, which MACRO uses to process programmed requests.

See the *RT–11 System Internals Manual* for more information on the internal data structure of a library file.

Accessing Library Files

You access object modules in a library file from another program by making calls or references to their global symbols. You then link the object modules with the program that uses them, producing a single load module (see Chapter 15 for a description of the Link Utility).

Calling and Terminating LIBR

To call the RT–11 librarian from the system device, issue the command:

```
.R LIBR RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal when it is ready to accept a command line.

Press CTRL/C twice to terminate the librarian at any time (or press CTRL/C once to terminate the librarian when it is waiting for terminal input) and return control to the monitor.

To restart the librarian, issue the following command in response to the monitor's dot prompt:

```
.R LIBR RET
```

or

```
.REENTER RET
```

Command-Line Syntax

Specify the LIBR command string in the following general format:

library-filespec[n],list-filespec[n]=input-filespecs/option

where:

library-filespec[n]	specifies the library file to be created or updated. The optional argument n specifies the number of blocks to allocate for the output file.
list-filespec[n]	specifies a listing file for the library's contents. The optional argument n specifies the number of blocks to allocate for the listing file.
input-filespecs	specifies the input object modules (you can specify up to six input files); it can also specify a library file to be updated.
option	specifies an option from Table 14–2.

Default File Types

You specify devices and file names in the standard RT–11 command string syntax, with default file types for object libraries assigned as follows:

Object File	Default File Type
List file	LST
Library output file	OBJ
Input file (library or module)	OBJ

If you do not specify a device, DK is the default device.

Input Files and Object Modules

Each LIBR input file consists of one or more object modules and is stored on a given device under a specific file name and file type. Once you insert an object module into a library file, you no longer reference the module by the name of the file of which it was a part; instead you reference it by its individual module name.

You assign the module name with the assembler through one of the following:

- A .TITLE statement in the assembly source program
- The default name .MAIN. in the absence of a .TITLE statement
- The subprogram name for FORTRAN routines

For example, an input file FORT.OBJ on DU1 can contain an object module ABC. When you have inserted the module into a library file, refer only to ABC (not FORT.OBJ).

Input files normally do not contain main programs, but rather subprograms, functions, and subroutines. The library file must never contain a FORTRAN BLOCK DATA subprogram because there is no undefined global symbol to cause the linker to load it automatically.

Creating an Object Library File

To create a library file, specify a file name on the output side of the command line.

The following example creates a new library called NEWLIB.OBJ on default device DK. The modules that make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device:

```
NEWLIB=FIRST,SECOND
```

Inserting Modules into a Library

Whenever you specify an input file without specifying an associated option, the librarian inserts the input file's modules into the library file you name on the output side of the command line. You can specify any number of input files.

If you include section names (by using /P) in the global symbol table and if you attempt to insert a file that contains a global symbol or PSECT (or CSECT) having the same name as a global symbol or PSECT already existing in the library file, the librarian displays a warning message (see the Creating Multiple Definition Libraries (/X) section of this chapter). The librarian updates the library file, ignores the global symbol or section name in error, and returns control to the CSI interpreter. You can then enter another command line.

Although you can insert object modules that exist under the same name (as assigned by the TITLE statement), this practice is not recommended because of possible confusion when you need to update these modules (see discussions on Replace Option (/R) and Update Option (/U)).

The librarian performs module insertion, replacement, deletion, merge, and update when creating the library file. Therefore, you must indicate the library file to which the operation is directed on both the input and output sides of the command line. Because the librarian creates a new output library file whenever it performs one of these operations, you must specify the library file first in the input field.

The following command line inserts the modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DU1: into a library file named DXYNEW.OBJ on the default device. The resulting library also includes the contents of library DXY.OBJ:

```
DXYNEW=DXY,DU1:FA,FB,FC
```

The next command line inserts the modules contained in files THIRD.OBJ and FOURTH.OBJ into the library NEWLIB.OBJ:

```
NEWLIB,LIST=NEWLIB,THIRD,FOURTH
```

The resulting library contains the original library, plus some new modules, and replaces the original library because the same name was used in this example for both input and output libraries.

Merging Library Files

You can merge two or more library files under one file name by specifying in a single command line all the library files to be merged. The librarian does not delete the individual library files following the merge unless the output file name is identical to one of the input file names.

The command syntax is as follows:

library-filespec=input-filespecs

where:

library-filespec	specifies the library file that will contain all the merged files. (If a library file already exists under this name, you must also indicate it in the input side of the command line so that it is included in the merge.)
input-filespec	specifies library files to be merged.

The following command combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK. Note that this replaces the old contents of MAIN.OBJ:

```
MAIN=MAIN,TRIG,STP,BAC
```

The next command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the file name FORT.OBJ:

```
FORT=A,B,C
```

NOTE

Library files that you combine, using PIP, are invalid as input to both the librarian and the linker.

Listing the Directory of a Library File

You can request a listing of the contents of a library file (the global symbol table) by indicating both the library file and a list file in the command line. Since a library file is not being created or updated, you do not need to indicate the file name on the output side of the command line; however, you must use a comma to designate a null output library file.

The command syntax is as follows:

,LP:=library-filespec

or

,list-filespec=library-filespec

where:

library-filespec	specifies the existing library file.
LP:	indicates that the listing is to be sent directly to the printer (or terminal, if you use TT).
list-filespec	specifies a listing file of the library file's contents.

The following command outputs to DU1, as LIST.LST a listing of all modules in the library file LIBFIL.OBJ, which is stored on the default device:

```
,DU1:LIST=LIBFIL
```

The next command sends to the printer a listing of all modules in the library file FLIB.OBJ, which is stored on the default device:

```
,LP:=FLIB
```

Here is a sample section of a large directory listing:

```
,TT:=SYSLIB
RT-11 LIBRARIAN V05.06  TUE 06-NOV-90 21:01:01
DK:SYSLIB.OBJ           TUE 06-NOV-90 20:59:47

MODULE          GLOBALS         GLOBALS         GLOBALS

                DCO$            ECO$            FCO$
+               GCO$            RCI$
                DIC$IS          DIC$MS          DIC$PS
+               DIC$SS          $DIVC           $DVC
                ADD$IS          ADD$MS          ADD$PS
+               ADD$SS          SUD$IS          SUD$MS
+               SUD$PS          SUD$SS          $ADD
```

The first line of the listing file shows the version of the librarian that was used and the current date and time. The second line prints the library file name and the date and time the library was created. Each line in the rest of the listing shows only the globals that appear in a particular module. If a module contains more global symbol names than can print on one line, a new line will be started with a plus (+) sign in column 1 to indicate continuation.

If you request a listing of a library file that was created with the /X or /N option, the listing includes module names under the MODULE heading.

LIBR Object-Library Option Descriptions

All Global and Absolute-Global Symbols (/A)

Use the /A option when you want all the global symbols to appear in the library file's directory. When you use /A, the librarian includes in the directory all absolute global symbols, including those that have a value of 0.

Normally, the librarian includes in the directory only global entry points (labels), but not absolute global symbols.

The following example places all the global symbols from modules MOD1 and MOD2 in the library directory for ALIB.OBJ:

```
ALIB=MOD1,MOD2/A
```

Command Continuation (/C or //)

You must use a continuation option whenever there is not enough room to enter a command on one line. The maximum number of input files that you can enter on one line is six; you can use the /C option or the // option to enter more.

Type the /C option at the end of the current line and repeat it at the end of subsequent command lines as often as necessary, so long as memory is available; if you exceed memory, an error message displays. Each continuation line after the first command line can contain only input file specifications (and no other options). Do not specify a /C option on the last line of input. If you use the // option, type it at the end of the first input line and again at the end of the last input line.

The command line in the following example creates library file ALIB.OBJ on default device DK; also on DK, it creates LIBLST.LST, a listing of the library file's contents. The file names of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DU1:

```
ALIB,LIBLST=DU1:MAIN,TEST,FXN/C
DU1:TRACK
```

The command line in the next example creates library file BLIB.OBJ on default device DK. It does not produce a listing. Input files are MAIN.OBJ from the default device, TEST.OBJ from DL1, FXN.OBJ from DL0, and TRACK.OBJ from DU1:

```
BLIB=MAIN//
DL1:TEST
DL0:FXN
DU1:TRACK//
```

Another version of this command line is:

```
BLIB=MAIN,DL1:TEST,DL0:FXN//
DU1:TRACK
//
```

LIBR Object-Library Option Descriptions

Delete (/D)

The /D option deletes modules and all their associated global symbols from a library file's directory. Since modules are deleted only from the directory (and not from the object module itself), all modules that were previously deleted are restored whenever you update that library, unless you use /D again to delete them.

When you use the /D option, the librarian prompts:

```
Module name?
```

Enter the name of the module to be deleted, then press RETURN. Continue entering all modules to be deleted, then press RETURN to the *Module name?* message to terminate input and initiate execution of the command line.

In the following example, the modules SGN and TAN are deleted from the library file TRAP.OBJ on DU1:

```
DU1:TRAP=DU1:TRAP/D  RET
Module name? SGN  RET
Module name? TAN  RET
Module name?  RET
```

In the next example the module FIRST is deleted from the library LIBFIL.OBJ; all modules in the file ABC.OBJ replace old modules of the same name in the library. It also inserts the modules in the file DEF.OBJ into the library:

```
LIBFIL=LIBFIL/D,ABC/R,DEF  RET
Module name? FIRST  RET
Module name?  RET
```

In the following example the librarian deletes two modules having the same name from the library file LIBFIL.OBJ:

```
LIBFIL=LIBFIL/D  RET
Module name? X  RET
Module name? X  RET
Module name?  RET
```

Extract (/E)

Using the /E option enables you to extract an object module from a library file and place it in an .OBJ file.

When you specify the /E option, the librarian displays:

```
Global?
```

Enter the name of the object module you want to extract. If you specify a global name, the librarian extracts the entire module of which that global is a part. Press RETURN to terminate the prompts for a global.

You cannot use the /E option on the same command line with another option.

The command line in the following example extracts the ATAN routine from the FORTRAN library, SYSLIB.OBJ, and stores it in a file called ATAN.OBJ on DX1:

```
DX1:ATAN=SYSLIB/E RET
Global? ATAN RET
Global? RET
```

In the next example the $PRINT routine is extracted from SYSLIB.OBJ and stored on DM1 as PRINT.OBJ:

```
DM1:PRINT=SYSLIB/E RET
Global? $PRINT RET
Global? RET
```

Delete Global (/G)

The /G option lets you delete specific global symbols from a library file's directory.

When you use the /G option, the librarian displays:

```
Global?
```

Enter the name of the global symbol you want to delete, then press RETURN. Continue until you have entered all globals to be deleted. Press RETURN to the *Global?* message to terminate input and execute the command line.

The following command instructs LIBR to delete the global symbols NAMEA and NAMEB from the directory found in the library file ROLL.OBJ on DK:

```
ROLL=ROLL/G RET
Global? NAMEA RET
Global? NAMEB RET
Global? RET
```

The librarian deletes globals only from the directory (and not from the library itself). Whenever you update a library file, all globals that you previously deleted are restored unless you use the /G option again to delete them. This feature lets you recover if you delete the wrong global.

Module Names (/N)

When you use the /N option on the first line of the command, the librarian includes module names in the directory. The linker loads modules from libraries based on undefined globals, not on module names. The linker also provides equivalent functions by using global symbols and not module names. Normally, then, it is a waste of space and a performance compromise to include module names in the directory.

If you do not include module names in the directory, the MODULE column of the directory listing is blank, unless the module requires a continuation line to display all its globals. A plus (+) sign in the MODULE column indicates continued lines. The /N option is useful mainly when you create a temporary library in order to obtain a directory listing.

If the library does not have module names in its directory, you must create a new library to include the module names. The following example illustrates how to do this. It creates a temporary new library from the current library (by specifying the null device for output) and lists its directory on the terminal. The current library OLDLIB remains unchanged:

```
NL:TEMP,TT:=OLDLIB/N
RT-11 LIBRARIAN V05.06  MON 04-MAR-91 20:36:41
NL:TEMP.OBJ             MON 04-MAR-91 20:36:40

MODULE          GLOBALS          GLOBALS         GLOBALS

IRAD50          IRAD50           RAD50
JMUL            JMUL
LEN             LEN
SUBSTR          SUBSTR
JADD            JADD
JCMP            JCMP
```

PSECT Names (/P)

The librarian does not include program-section (PSECT) names in the directory unless you use the /P option on the first line of the command. The linker does not use section names to load routines from libraries—in fact, including the names can decrease linker performance. Including program section names also causes a conflict in the library directory and subsequent searches, since the librarian treats section names and global symbols identically.

This option is provided for compatibility with RT–11 V2C. Digital recommends that you avoid using it with later versions of RT–11.

Replace (/R)

Use the /R option to replace modules in a library file. The /R option replaces existing modules in the library file you specify as output with the modules of the same names contained in the file(s) you specify as input. In the command line, enter the input library file before the files used in the replacement operation.

If an old module does not exist under the same name as an input module, or if you specify the /R option on a library file, the librarian displays an error message followed by the module name and ignores the replace command.

The /R option must follow each input file name containing modules for replacement.

The following command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added to TFIL. All files are stored on the default device DK:

```
TFIL=TFIL,INA,INB/R,INC
```

The same operation occurs in the next command, except that this updated library file is assigned the new name XFIL:

```
XFIL=TFIL,INA,INB/R,INC
```

Update (/U)

The /U option lets you update a library file by combining the insert and replace functions. If the object modules that compose an input file in the command line already exist in the library file, the librarian replaces the old modules in the library file with the new modules in the input file. If the object modules do not already exist in the library file, the librarian inserts those modules into the library. (Note that some of the error messages that might occur with separate insert and replace functions do not display when you use the update function.)

/U must follow each input file that contains modules to be updated. Specify the input library file before the input files in the command line.

The following command line instructs the librarian to update the library file BALIB.OBJ on the default device. First, the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under the same names, the modules are inserted. The modules from the file TAL.OBJ are then inserted; an error message displays if the name of a module in TAL.OBJ already exists. See the Combining Object-Library Options section for a complete description of the order in which the librarian does tasks:

```
BALIB=BALIB,FOLT/U,TAL,BART/U
```

In the next example, two object modules have the same name, X, in both Z and XLIB. First, both are deleted from XLIB so that both modules called X in file Z are correctly placed in the library. Globals SEC1 and SEC2 are also deleted from the directory, but automatically return the next time the library XLIB.OBJ is updated:

```
XLIB=XLIB/D,Z/U/G [RET]
Module name? X [RET]
Module name? X [RET]
Module name? [RET]
Global? SEC1 [RET]
Global? SEC2 [RET]
Global? [RET]
```

Wide (/W)

The /W option gives you a wider listing if you request a listing file. The wider listing has six global columns instead of three, as in the normal listing. This is useful if you list the directory on a printer or a terminal set to 132 columns.

Multiple Definition Library (/X)

The /X option lets you create libraries that can have more than one definition for a global entry point. These libraries are called multiple definition libraries. They are processed differently from libraries that contain only one definition for each global entry-point name that appears in the library's directory (for more information on processing multiple definition libraries, see Chapter 15).

In multiple definition libraries, two library modules may use the same global entry-point name, and both definitions may appear in the entry-point table (EPT). At least one entry-point name should be unique in each module so that you can easily identify it.

When you use the /X option, the librarian does not issue the *?LIBR-W-Invalid* insert of AAAAAA message when it encounters a duplicate global symbol name, and the global name will appear in the directory for each module that defines it. In addition, the /X option causes the librarian to turn on the /N option (see the Module Names (/N) option description).

The following example shows the creation of the multiple definition library MLTLIB from modules MOD1, MOD2, and MOD3, and lists the library on the terminal. Since MOD3 contains only absolute global symbols, this example must also use the /A option:

```
*MLTLIB,TT:=MOD1,MOD2,MOD3/X/A

RT-11 LIBRARIAN V05.06    THU 15-NOV-90 09:45:31
DK:MLTLIB.OBJ             THU 15-NOV-90 09:45:31
MODULE      GLOBALS      GLOBALS     GLOBALS

MOD1        OMA$R        SWP$        ATP$
MOD2        ATP$         OMA$R       MER$CR
            LBM
MOD3        ATP$         OMA$R       MER$CR
            ENTZ
```

Combining LIBR Options

You can specify two or more library functions in the same command line, except for the /E and /M options. LIBR performs functions (and issues appropriate prompts) in the following order:

1. /C or //
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

Here is an example that combines options:

```
FILE,LP:=FILE/D,MODX,MODY/R
Module name? XYZ
Module name? A
Module name?
```

The librarian performs the functions in this example in the following order:

1. Deletes modules XYZ and A from the library file FILE.OBJ.
2. Replaces any duplicate of the modules in the file MODY.OBJ.
3. Inserts the modules in the file MODX.OBJ.
4. Lists the directory of FILE.OBJ on the printer.

Creating Macro Libraries

The librarian lets you create MACRO–11 libraries for the V03 or later MACRO–11 assembler. This reduces macro search time.

These are some main points concerning MACRO–11 libraries:

- The .MACRO directive produces the entries (macro names) in the library directory.
- LIBR does not maintain a directory file listing macro libraries. However, you can display the ASCII input file to list the macros in the library.
- The default input file type for macro files is MAC. The default output file type for macro library files is MLB.
- If you give the library file the same name as one of the input files, the librarian displays the error message: *?LIBR-F-Output and input filenames the same*.
- The librarian removes all comments from your source input file except for those within a macro (that is, between a .MACRO and .ENDM pair of directives). Because comments take up space during the assembly and in the library, remove them from macros wherever possible before creating a macro library (if you want to save space and shorten assembly time).

Options for Creating Macro Libraries

Table 14–1 summarizes the options you can use with macro libraries. The options are explained in detail in the following two sections.

Table 14–1: LIBR Macro Options

Option	Command Line	Meaning
/C	Any but last	Command continuation; allows you to type the input specification on more than one line.
/M[:n]	First	Macro; creates a macro library from the ASCII input file containing .MACRO directives.
//	First and last	Command continuation; allows you to type the input specification on more than one line.

LIBR Macro-Library Option Descriptions

Command Continuation (/C or //)

These options are the same for both macro libraries and object libraries.

Macro (/M[:value])

The /M[:value] option creates a macro library file from an ASCII input file that contains .MACRO directives. The optional *value* argument determines the amount of space to allocate for the macro name directory by specifying the number of macros you want the directory to hold. Remember that *value* is interpreted as an octal number; you must follow *value* with a decimal point (*value.*) to indicate a decimal number. One block of library directory space holds 64 macros. The default value for *value* is 128, enough space for 128 macros, which will use 2 blocks for the macro name table.

The command-line syntax is as follows:

library-filespec=input-filespec/M[:value]

where:

library-filespec	specifies the macro library to be created.
input-filespec	specifies the ASCII input file that contains .MACRO definitions.

The continuation options (/C or //) are the only options you can use with the macro option.

The following example creates the macro library SYSMAC.SML from the ASCII input file SYSMAC.MAC. Both files are on device DK:

```
SYSMAC.SML=SYSMAC/M
```

LIBR Option Summary

Table 14–2 summarizes the options and functions available for use with LIBR.

Table 14–2: LIBR Option Summary

Option	Command Line	Function
/A	First	Puts all globals in the directory, including all absolute global symbols.
/C	Any but last	Command continuation; allows you to type the input specification on more than one line.
/D	First	Delete; deletes modules that you specify from a library file.
/E	First	Extract; extracts a module from a library and stores it in an OBJ file.
/G	First	Global deletion; deletes global symbols that you specify from the library directory.
/M[:value]	First	Creates a macro library.
/N	First	Names; includes the module names in the directory.
/P	First	P-sect names; includes the program section names in the directory.
/R	First	Replace; replaces modules in a library file. This option must follow the file specification to which it applies.
/U	First	Update; inserts and replaces modules in a library file. This option must follow the file specification to which it applies.
/W	First	Indicates a wide format for the listing file.
/X	First	Enables multiple definitions of global entry points to appear in the library entry point table.
//	First and last	Command continuation; lets you type the input specification on more than one line.

There is no option to indicate module insertion. If you do not specify an option, the librarian automatically inserts modules into the library file.

DCL Equivalents of LIBR Operations

Table 14–3 lists the DCL commands that are equivalent to LIBR utility operations.

Table 14–3: DCL Equivalents of LIBR Utility Operations

CSI Option	DCL Command(s)	DCL Option(s)
/A	–	
/C	LIBRARY	/PROMPT
/D	LIBRARY	/DELETE
/E	LIBRARY	/EXTRACT
/G	LIBRARY	/REMOVE
/M[:value]	LIBRARY	/MACRO[:value]
/N	–	
/P	–	
/R	LIBRARY	/REPLACE
/U	LIBRARY	/UPDATE
/W	–	
/X	–	
//	–	

Chapter 15
The Linker Utility (LINK)

The Linker Utility (LINK) converts *object modules* into *load modules*.

An object module is the primary output of an assembler or compiler, which can be linked with other modules and loaded into memory as a runnable program. The object module is composed of the relocatable machine-language code, relocation information, and the corresponding global-symbol table defining the use of the symbols within the program.

A global symbol is a symbol representing a value (constant or variable) or a label (to instructions or to data). The symbol is *global* in that it is used in a program module outside of the one in which it is defined. In contrast, a *local* symbol can be used only in the module in which it is defined.

A load module is a program in a format ready for loading into memory and executing.

See the *Introduction to RT–11* for an introductory-level description of the linking process. See the *RT–11 System Internals Manual* for an in-depth description of the components of the RT–11 operating system and how it uses memory.

Functions of the Linker

When the linker processes object modules, it does the following:

- Joins together (links) object modules that use global symbols with the object module that defines the symbol.
- Searches the *library files* you specify to locate global symbols not defined in your specified object modules. A library file is a file containing one or more relocatable object modules, which are routines that can be incorporated into programs. See Chapter 14 describing the LIBR utility for more information on library files.
- Produces a symbol-table definition file, if specified.
- Relocates your program module (and individual object modules) as necessary and assigns absolute memory addresses.
- Creates an overlay structure, if specified, and includes the necessary run-time overlay handlers and tables. The linker can overlay:
  - — Low memory (physical memory from 0 to 28K words)
  - — Extended memory (physical memory above the 28K word boundary)
- Allows you to link programs with separated I (instruction) and D (data) space. These programs must be first written with separated I and D space and then

can be run only under the ZB or ZM monitor. Writing a program that separates instruction and data space allows that program to run more quickly.

- Produces a load module, that is, the initial control block for the linked program that the GET, R, RUN, SRUN, FRUN and VRUN commands use.
- Produces a load map, if specified, that shows the layout of the load module.

Function Order

The linker requires two passes over the input modules.

1. During the first pass:
   - — The LINKER constructs the symbol table, which includes the names of all the program sections and global symbols in the input modules. A program section or PSECT is a named, contiguous unit of code (instructions or data) that is considered an entity and that can be relocated separately without destroying the logic of the program.
   - — Next, the linker scans the library files to resolve undefined global symbols; it links only those modules that are required to resolve undefined global symbols.
2. During the second pass over the input modules, the linker reads in referenced object modules, performs most of the functions in the preceding list, and produces the load module.

Calling and Terminating the Linker

To call the linker from the system device, respond to the monitor dot prompt by typing:

```
.R LINK RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal when it is ready to accept a command line. If you only press RETURN at this point, the linker displays its current version number.

Press CTRL/C twice to terminate the linker at any time (or press CTRL/C once to terminate the linker when it is waiting for terminal input) and return control to the monitor. To restart the linker, type R LINK or REENTER in response to the monitor's dot prompt.

Command-Line Syntax

Linker input and output is in the form of modules. The linker uses one or more input modules to produce a single output (load) module.

The linker accepts object modules, library modules, and symbol-table definition files as input. The linker produces load modules, a load map, and symbol-table definition file as output.

Though the linker accepts input from any random-access volume on the system, there must be at least one random-access volume (disk, diskette) for memory-image or relocatable-format output.

Enter first command string in response to the linker's prompt in this syntax:

[binary-file],[map-file],[symbol-file]=object-file[/option...][,...object-file[/option...]]

where:

binary-file	specifies the device, file name, and file type to be assigned to the linker's output load-module file
map-file	specifies the device, file name, and file type of the load-map output file
symbol-file	specifies the device, file name, and file type of the symbol-definition file
object-file	specifies an object module, a library file, or a symbol-table file, created in a previous link
/option	is one of the options listed in Table 15–6

In each of the preceding file specifications, the device should be a random-access device, with these two exceptions:

The output device for the load-map file
The output device for an LDA file (if you use the /L option)

The device for the two exceptions can be any RT–11 device. If you do not specify a device, the linker uses the default device (DK).

Note that the linker load map contains lowercase characters. Use the SET LP LC command to enable lowercase printing if your printer has lowercase characters.

If you do not specify an output file, the linker assumes that you do not want the associated output. For example, if you do not specify the load module and load map (by using a comma in place of each file specification) the linker displays only error messages, if any occur.

Default Devices and File Types

Specification	Device	File Name	File Type
Load Module	DK	None	SAV, REL(/R), LDA(/L)
Load Map	DK or same as load module	None	MAP
Symbol Definition Output	DK or same as previous output device	None	STB
Object Module	DK or same as previous object module	None	OBJ

If you make a syntax error in a command string, RT–11 displays an error message. You can then retype the new command string following the asterisk. Similarly, if you specify a nonexistent file, a warning message occurs, control returns to the CSI interpreter, the asterisk prompt is displayed, and you can reenter the command string.

Linker Prompts

Some of the linker operations prompt for more information, such as the names of specific global symbols or sections. The linker issues the prompt after you have entered all the input specifications, but before the actual linking begins. Table 15–1 shows the sequence in which the prompts occur.

Table 15–1: Linker Prompting Sequence

Prompt	Option
Transfer symbol?	/T
Stack symbol?	/M
Extend section? Extend instruction section? Extend data section?	/E:value[:type]
Boundary section? Instruction boundary section? Data boundary section?	/Y:value[:type]
Round section? Round instruction section? Round data section?	/U:value[:type]
Load section:address?	/Q
Library search?	/I
Duplicate symbol?	/D

The library search, load section, and duplicate symbol prompts can accept more than one symbol and are terminated by pressing RETURN in response to the prompt.

Note that if the command lines are in a command file and the linker encounters an end-of-file before the prompting information has been supplied, the linker displays the prompt messages on the terminal.

The following example shows how the linker prompts for information when you combine options:

```
*LK001=LK001/T/M/E:100/Y:400/U:20/I/Q/D RET
Transfer symbol? O.ODT RET
Stack symbol? ST3 RET
Extend section? CHAR RET
Boundary section? CODE RET
Round section? STKSP RET
Load section:address? MAIN:100000 RET
Load section:address? RET
Library search? $SHORT RET
Library search? RET
Duplicate symbol? RTN RET
Duplicate symbol? RET
*
```

Linker Input

This section further explains object and library modules.

Input Object Modules

Object files, consisting of one or more object modules, are the input to the linker. (Entering files that are not object modules may result in a fatal error.) Object modules are created by language translators such as the FORTRAN compiler and the MACRO–11 assembler. The module name item declares the name of the object module. (See the Output Load Map section.)

The first six Radix–50 characters of the .TITLE assembler directive are used as the name of the object module. These six characters must be Radix–50 characters (the linker ignores any characters beyond the sixth character). The linker displays the first module name it encounters in the input file stream (normally the main routine of the program) on the second line of the map following TITLE:. The linker also uses the first identity label (issued by the .IDENT directive) for the load map. It ignores additional module names.

The linker reads each object module twice. During the first pass it reads each object module to construct a global symbol table and to assign absolute values to the program section names and global symbols. The linker uses the library files to resolve undefined globals. It places their associated object modules in the root if the global symbols in the module are referenced from more than one overlay segment or from the root.

The *root segment* or root is the segment of an overlay structure that, when loaded, remains resident in memory during the execution of a program.

An *overlay segment* or overlay is a section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments when called from the root segment or another overlay segment.

If you use the /D option and the global symbols are not referenced from the root, the linker places a copy of the global symbols you specify in each segment that references them. (See the description of the /D option in the LINK Option Descriptions section.) On the second of its two passes, the linker reads the object modules, links and relocates the modules, and outputs the load module.

Symbol-table definition files are special object files that can serve as input to LINK anywhere other object files are allowed.

Input Library Modules

The RT–11 linker can automatically search libraries. Libraries consist of library files, which are specially formatted files produced by the librarian program (described in Chapter 14) that contain one or more object modules. The object modules provide routines and functions to aid you in meeting specific programming needs. (For example, FORTRAN has a set of modules containing all necessary computational functions—SQRT, SIN, COS, and so on.) You can use the librarian to create and

update libraries. Then you can easily access routines that you often use or routines that different programs use. Selected modules from the appropriate library file are linked as needed with your program to produce one load module. Libraries are further described in Chapter 14.

NOTE

Library files that you combine with the monitor COPY command or with the PIP /U or /B option (described in the *Peripheral Interchange Utility (PIP)* chapter in Part II of this manual) are invalid as input to both the linker and the librarian.

You specify libraries in a command string in the same way you specify normal modules; you can include them anywhere in the command string. If you are creating an overlay structure, specify libraries before you specify the overlay structure. Do not specify libraries on the same line as overlay segments. If a global symbol is undefined at the time the linker encounters the library in the input stream, and if a module is included in the library that contains that global definition, then the linker pulls that module from the library and links it into the load image. Only the modules needed to resolve references are pulled from the library; unreferenced modules are not linked.

Modules in one library can call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls Y from the BLIB library. To correctly resolve all globals, the order of ALIB and BLIB should appear in the command line as:

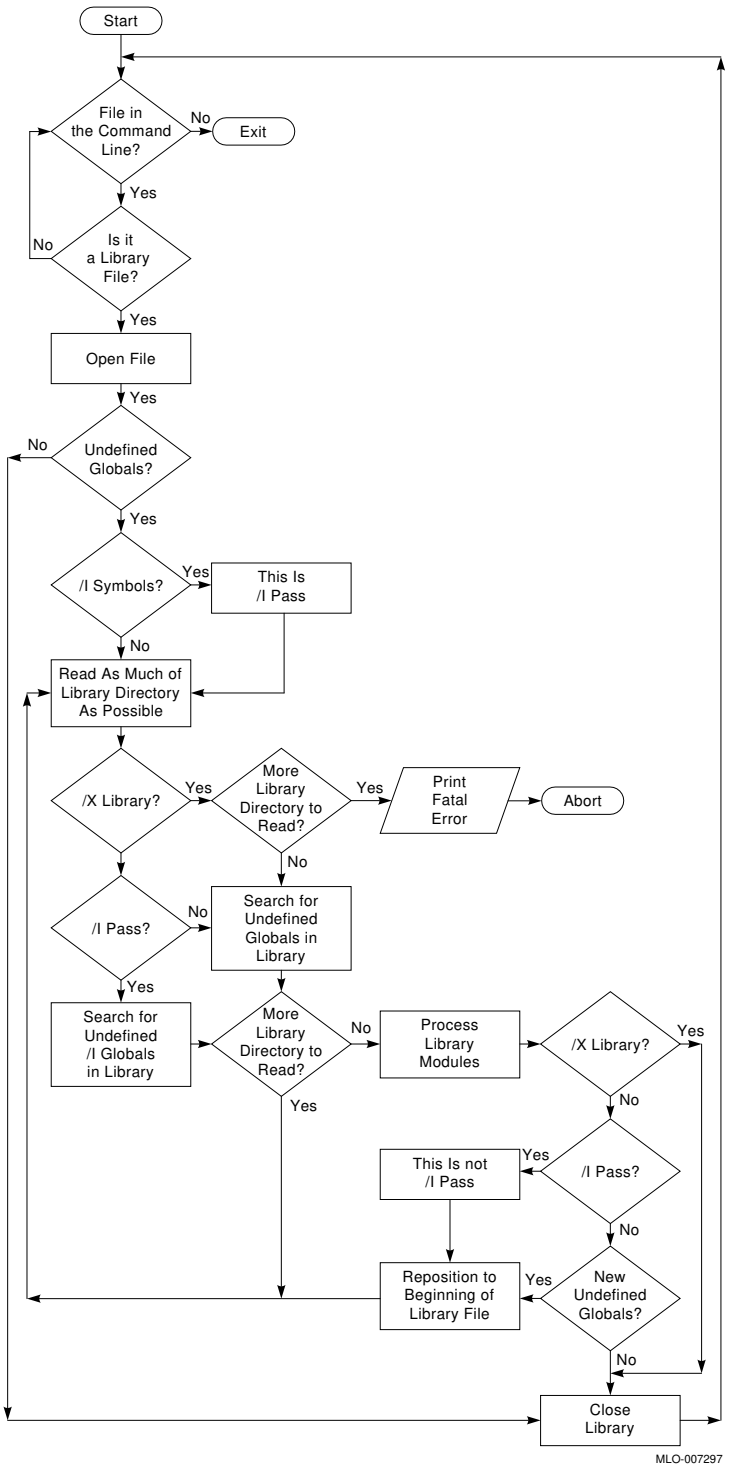
```
*Z=B,ALIB,BLIB
```

Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls Y, which is brought from BLIB into the root.

Library Module Processing

The linker selectively relocates and links object modules from specific user libraries that were built by the librarian. Figure 15–1 diagrams this general process. During pass 1 the linker processes the input files in the order in which they appear in the input command line. If the linker encounters a library file during pass 1, it takes note of the library in an internal save status block, and then proceeds to the next file. The linker processes only nonlibrary files during the initial phase of pass 1. In the final phase of pass 1, the linker processes only library files. This is when it resolves the undefined globals that were referenced by the nonlibrary files.

Figure 15–1: Library Searches



The linker processes library files in the order in which they appear in the input command line. The default system library (SY:SYSLIB.OBJ) is always processed last.

The search method the linker uses allows modules to appear in any order in the library. You can specify any number of libraries in a link and they can be positioned anywhere, with the exception of forward references between libraries, and they must come before the overlay structure. The default system library, SY:SYSLIB.OBJ, is the last library file the linker searches to resolve any remaining undefined globals.

Some languages, such as FORTRAN, have an Object Time System (OTS) that the linker takes from a library and includes in the final module. The most efficient way to accomplish this is to include these OTS routines (such as NHD, OTSCOM, and V2NS for FORTRAN) in SY:SYSLIB.OBJ. See the *RT–11 Installation Guide* for details on how to do this.

Libraries are input to the linker the same way as other input files. Here is a sample LINK command string:

```
*TASK01,LP:=MAIN,MEASUR
```

This causes program MAIN.OBJ to be read from DK as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The linker tries to satisfy any remaining undefined globals from the default library, SY:SYSLIB.OBJ. The load module, TASK01.SAV, is stored on DK and a load map prints on the printer.

Multiple-Definition Libraries

In addition to the libraries explained so far, LINK processes multiple-definition libraries. Digital does not recommend that you use this type of library in normal situations; its primary purpose is to provide special functions for RSTS. These libraries differ from other libraries in that they can contain more than one definition for a given global. You specify multiple-definition libraries in the command line the same way you specify normal libraries. Modules that LINK obtains from multiple-definition libraries always appear in the root.

It is useful to be aware of the differences between processing normal and multiple-definition libraries. When you include modules from a multiple-definition library, LINK has to store that library's directory in an internal buffer. A library's directory is often called an entry point table (EPT). If a library EPT is too large to fit into the internal buffer, LINK displays a message instructing you to use the /G option. The /G option changes the buffer's size to accommodate the largest EPT of all the multiple-definition libraries you are using. Use the /G option only when LINK indicates it is required.

When a global symbol from a multiple-definition library matches an undefined global, LINK removes from the undefined global list all other globals defined in the same library. LINK does this before it processes the library module. Thus, two modules with identical globals will not appear in the linked module.

NOTE

The order of modules in multiple-definition libraries is very important and will affect which modules LINK uses. The increased EPT size (due to duplicate entries, in addition to module name entries) will also slow LINK down.

LINK cannot locate in a library module (and therefore resolve) any absolute global symbol, unless the symbol was included in the module using the LIBR /A option or the symbol was associated with at least one relative global symbol. The LIBR /A option is generally not used because it forces all global and absolute global symbols into the library module directory, making the directory quite large. It is generally better to associate any absolute global symbol with at least one relative global symbol.

For example, assume that a library module contains the symbol $arger that has an absolute value of 23 ($ARGER==23). LIBR cannot locate and therefore LINK cannot resolve the symbol $ARGER. You should associate that absolute symbol with a relative symbol; for example, GEO (GEO::$ARGER==23). Then, if you specified the library containing GEO in your command line, LINK could process the following code:

```
.GLOBL  GEO,$ARGER
TRAP    $ARGER
```

Linker Output

The section further explains the load module and load map.

Output Load Module

The primary output of the linker is a load module that you can run under RT–11. The linker creates as a load module a memory-image file (file type of SAV) for use under a single-job unmapped system or as the background job under a multi-job unmapped system. Save images can also be run as virtual foreground jobs under a mapped monitor. If you need to execute a program in the foreground, use the /R option to produce a relocatable format (file type of REL) foreground load module. The linker can produce an absolute load module (file type of LDA) if you need to load the module with the Absolute Loader. See the *RT–11 Volume and File Formats Manual* for more details on these formats.

The load module for a memory-image file is arranged as:

```
Root Segment      Overlay Segments
                  (optional)
```

The load module for a relocatable-image file is arranged as follows:

```
Root Segment     Overlay Segments     Relocation information for root
                 (optional)           and overlay segments
```

The first 256-word block of the root segment (main program) contains the memory usage bitmap and the locations the linker uses to pass program control parameters. The memory usage bitmap outlines the blocks of memory that the load module uses; it resides in locations 360 through 377.

Table 15–2 lists the parameters that appear in the absolute block, the addresses the parameters occupy, and the conditions under which they are set.

Table 15–2: Absolute Block Parameters

Address	Parameter	When Set
0	Identification of a program that was created with /V option	Only with /V
2	Highest virtual memory address used by the program	Only with /V
14,16	(Mapped monitor only) BPT trap	Only with /R
20,22	(Mapped monitor only) IOT trap	Only with /R
34,36	TRAP vector	Only with /R
40	Start address of program	Always
42	Initial setting of SP (stack pointer)	Always
44	Job Status Word (overlay bit set by LINK)	Always

Table 15–2 (Cont.): Absolute Block Parameters

Address	Parameter	When Set
46	USR swap (set by user) address; (0 implies normal location)	Always
50	Highest memory address used by the program (high limit)	Always
52	Size of root segment in bytes	Only with /R
54	Stack size in bytes (value with /R or default 128)	Only with /R
56	Size of overlay region in bytes	Only with /R
60	Identification of a file in relocatable (.REL) format	Only with /R
62	Relative block number for start of relocation information	Only with /R
64	Start address of overlay table	With /O or /V
66	Start of virtual overlay segment information in overlay handler tables	Only with /V
360–377	Memory usage bitmap	Always, except with /X or /L

The linker stores default values in locations 40, 42, and 50, unless you use options to specify otherwise. The /T option affects location 40, for example, and /M affects location 42. You can also use the .ASECT directive to change the defaults. The overlay bit is located in the job status word. LINK automatically sets this bit if the program is overlaid. Otherwise, the linker initially sets location 44 to 0. Location 46 also contains zero unless you specify another value by using the .ASECT directive.

You can assign initial values to memory locations 0–476 (which include the interrupt vectors and system communication area) by using an .ASECT assembler directive. The values appear in block 0 of the load module, but there are restrictions on the use of .ASECT directives in this region. You should not modify location 54 or locations 360–377 because the memory usage map is passed in those locations. In addition, for foreground links, modifications of words 52–62 are not permitted because additional parameters are passed to the FRUN command in those locations.

You can use an .ASECT directive to set any location that is not restricted, but be careful if you change the system communication area. The program itself must initialize restricted areas, such as locations 360–377. There are no restrictions on .ASECT directives if the output format is LDA.

Locations in addresses 0–476 might not be loaded at execution time, even though your program uses an .ASECT to initialize them. For background programs, this is because the R, RUN, and GET commands do not load addresses that are protected by the monitor's memory protection map. For foreground programs, the FRUN command loads only locations 14–22 and 34–50 and ignores all other low-memory locations. To initialize a location at run time, use the .PROTECT programmed request. If it is successful, follow it with a MOV instruction to modify the location.

Output Load Map

The linker can produce a load map following the completion of the initial pass. This map, shown in Figure 15–2, diagrams the layout of memory for the load module.

The load map lists each program section that is included in the linking process. The line for a section includes the name and low address of the section and its size in bytes. The rest of the line lists the program section attributes, as shown in Table 15–4. The remaining columns contain the global symbols found in the section and their values.

Figure 15–2: Sample Load Map

```
1     RT-11   LINK    V06.01        Load Map          Friday 25-Jan-91 11:25      Page 1
2     TEST    .SAV        Title:    TEST      Ident:
3
4     Section    Addr     Size      Global    Value   Global   Value    Global    Value
5
6      . ABS.    000000   001000 = 256.       words   (RW,I,GBL,ABS,OVR)
7                001000   000200 =  64.       words   (RW,I,LCL,REL,CON)
8      TEST      001200   000174 =  62.       words   (RW,I,LCL,REL,CON)
9                                  START     001200  EXIT     001240
10
11     Transfer address = 001200, High limit = 001372 = 381.    words
```

The following table describes each line in the preceding sample load map.

Line	Contents
1	Load map header.
2	Program name, program title (.MAIN. default) and identity (default is blank).
4	PSECT description header. Section indicates the PSECT name; Addr indicates the PSECT start address; Size indicates PSECT length in octal bytes; Global and Value list the PSECT globals and their associated octal values.
6	Absolute PSECT, . ABS. This line includes the absolute PSECT's start address, length and attributes (for a complete description of these abbreviations, see Table 15–3). The linker always includes a . ABS. PSECT in the link.
7	Unnamed PSECT. This PSECT appears in the load map after the absolute PSECT. For overlaid programs, the unnamed PSECT appears in the load map after the overlay table PSECT. (See Appendix A).
8–9	TEST PSECT. Line 9 lists TEST's two globals, START and EXIT, with their associated values.
11	Transfer address indicates the address in memory where the program starts. High limit indicates the last address used by the program. The number of words in the program appears last.

The map begins with the linker version number, followed by the date and time the program was linked. The second line lists the file name of the program, its title (which is determined by the first module name record in the input file), and the first identification record found. The absolute section is always shown first, followed by any nonrelocatable symbols. The modules located in the root segment of the load

module are listed next, followed by those modules that were assigned to overlays in order by their region number. (See the Creating an Overlay Structure section in Appendix A.) Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code in both octal bytes and decimal words. If you use the /N option, a cross-reference of all global symbols defined during the linking process follows the transfer address line. See Appendix A for a sample and description of a global cross-references table.

NOTE

The load map does not reflect the absolute addresses for a REL file that you create to run as a foreground job; you must add the base relocation address determined at FRUN time to obtain the absolute addresses. The linker assumes a base address of 1000.

For example, assume the FRUN command is used to run the program TEST:

```
.FRUN TEST/P
Loaded at 127276
```

The /P option causes FRUN to print the load address, which is 127276 in this example. To calculate the actual location in memory of any global in the program, first subtract 1000 from that global's value. (The value 1000 specifies the base address assigned by the linker. This offset is not used at load time.) Then add the result to the load address determined with /P. The final result specifies the absolute location of the global.

How the Linker Structures the Load Module

When the linker processes assembled or compiled object modules, it creates a load module in which it has:

- Assigned all absolute addresses
- Created an absolute section
- Allocated memory for the program sections

Assigning Absolute Addresses

See the *Introduction to RT–11* for an explanation of why and how the linker assigns absolute addresses.

Creating an Absolute Section

The absolute section is often called the ASECT because the .ASECT assembler directive allows information to be stored there. The absolute section appears in the load map with the name .ABS and is always the first section in the listing. The absolute section typically ends at address 1000_8 and contains the following:

— System communication area

— Hardware vectors

— User stack

The system communication area resides in locations 0–377, and contains data the linker uses to pass program control parameters and a memory usage bitmap. The Output Load Module section provides a detailed description of each location in the system communication area.

The stack is an area that a program can use for temporary storage and subroutine linkage. General register 6, the stack pointer (SP), references the stack.

Allocating Memory for Program Sections

See the *Introduction to RT–11* for an introductory explanation of how LINK allocates memory. See Appendix A for a detailed explanation of how LINK uses overlays when allocating memory.

The load module the linker produces contains an absolute section followed by program sections. The program section or PSECT is a program's basic unit of memory. The linker allocates memory by PSECTs.

The set of attributes associated with each PSECT controls the allocation and placement of the section within the load module. The PSECT, as the basic unit of memory for a program, has:

- A name by which it can be referenced

- A set of attributes that define its contents, mode of access, allocation, and placement in memory
- A length that determines how much storage is reserved for the PSECT

You create PSECTs by using a COMMON statement in FORTRAN, or the .PSECT (or .CSECT) directive in MACRO. You can use the .PSECT (or .CSECT) directive to attach attributes to the section. Note that the attributes that follow the PSECT name in the load map are not part of the name; only the name itself distinguishes one PSECT from another. You should make sure, then, that PSECTs of the same name that you want to link together also have the same attribute list. If the linker encounters PSECTs with the same name that have different attributes, it displays a warning message and uses the attributes from the first time it encountered the PSECT.

PSECT Attributes

The linker collects from the input modules scattered references to a PSECT and combines them in a single area of the load module. The attributes, which are listed in Table 15–3, control the way the linker collects and places this unit of storage.

Table 15–3: PSECT Attributes

Attribute	Value	Explanation
Access-code *	RW (Read/Write)	Data can be read from, and written into, the PSECT.
	RO (Read Only)	Data can be read from, but cannot be written into, the PSECT.
Type-code	D (Data)	The PSECT contains data, concatenated by byte.
	I (Instruction)	The PSECT contains either instructions, or data and instructions, concatenated by word.
Scope-code	GBL (Global)	The PSECT name is recognized across segment boundaries. The linker allocates storage in the root for the PSECT from references outside the defining overlay segment. If the PSECT is referenced only in one segment, that PSECT has space allocated in that segment only.
	LCL (Local)	The PSECT name is recognized only within each individual segment. The linker allocates storage for the PSECT from references within the segment only.
	SAV (Save)	The PSECT name is recognized across segment boundaries. The linker always allocates storage in the root for the PSECT.

*Not supported

Table 15–3 (Cont.): PSECT Attributes

Attribute	Value	Explanation
Reloc-code	REL (Relocatable)	The base address of the PSECT is relocated relative to the virtual base address of the program.
	ABS (Absolute)	The base address of the PSECT is not relocated. It is always 0.
Alloc-code	CON (Concatenate)	All allocations to a given PSECT name are concatenated. The total allocation is the sum of the individual allocations.
	OVR (Overlay)	All allocations to a given PSECT name overlay each other. The total allocation is the length of the longest individual allocation.

The Scope-Code, Alloc-Code, and Type-Code Attributes

- **Scope-Code**

  The scope-code is meaningful only when you define an overlay structure for the program. In an overlaid program, a global section is known throughout the entire program. Object modules contribute to only one global section of the same name. If two or more segments contribute to a global section, then the linker allocates that global section to the root segment of the program.

  In contrast to global sections, local sections are only known within a particular program segment. Because of this, several local sections of the same name can appear in different segments. Thus, several object modules contributing to a local section do so only within each segment. An example of a global section is named COMMON in FORTRAN. An example of a local section is the default blank section for each macro routine.

- **Alloc-Code**

  The alloc-code determines the starting address and length of memory allocated by modules referencing a common PSECT.

  If the alloc-code indicates that a common PSECT is to be overlaid, the linker stores the allocations from each module, starting at the same location in memory. The linker determines the total size from the length of the longest reference to the PSECT. Each module's allocation of memory to a location overwrites that of a previous module.

  If the alloc-code indicates that a PSECT is to be concatenated, the linker places the allocations from the modules one after the other in the load module; it determines the total allocation from the sum of the lengths of the contributions.

- **Type-Code**

  The allocation of memory for a PSECT always begins on a word boundary. If the PSECT has the D (data) and CON (concatenate) attributes, all storage that

subsequent modules contribute is appended to the last byte of the previous allocation. This occurs whether or not that byte is on a word boundary. For a PSECT with the I (instruction) and CON attributes, however, all storage that subsequent modules contribute begins at the nearest following word boundary.

Any data (D) PSECT that contains references to word labels must start on a word boundary. You can do this by using the .EVEN assembler directive at the end of each module's concatenated PSECT. (If this is not done, the program may fail to link, displaying the message *?LINK-F-Word relocation error in FILNAM*.

Special Cases of PSECTs

The .CSECT directive of MACRO is converted internally by both MACRO and the linker to an equivalent .PSECT with fixed attributes. An unnamed CSECT (blank section) is the same as a blank PSECT with the attributes RW, I, LCL, REL, and CON.

A named CSECT is equivalent to a named PSECT with the attributes RW, I, GBL, REL, and OVR. Table 15–4 shows these sections and their attributes.

Table 15–4: Section Attributes

Section	Access-Code	Type-Code	Scope-Code	Reloc-Code	Alloc-Code
.CSECT	RW	I	LCL	REL	CON
.CSECT name	RW	I	GBL	REL	OVR
.ASECT (.ABS.)	RW	I	GBL	ABS	OVR
COMMON/name/	RW	D	GBL	REL	OVR
vsect (.VIR.)	RW	D	GBL	REL	CON

The names assigned to PSECTs are not considered to be global symbols; you cannot reference them as such. For example, consider the following statement:

```
MOV     #PNAME,R0
```

This statement, where PNAME is the name of a section, is invalid and generates the undefined global error message if no global symbol of PNAME exists. A name can be the same for both a PSECT name and a global symbol. The linker treats them separately.

How PSECTs Are Arranged in a Load Module

The linker determines the memory allocation of PSECTs by the order of occurrence of the PSECTs in the input modules. Table 15–5 shows the order in which PSECTs appear for both overlaid and nonoverlaid files.

Table 15–5: PSECT Order

Nonoverlaid File	Overlaid File
Absolute (. ABS)	Absolute (. ABS)
Blank	Overlay handler ($OHAND)
Named (NAME)	Overlay table ($OTABL)
	Blank
	Named (NAME)

If there is more than one named section, the named sections appear in the order in which they occur in the input files. For example, the FORTRAN compiler arranges the PSECTs in the main program module so that the USR can swap over pure code in low memory rather than over data required by the function making the USR call.

If the size of the blank PSECT is 0, it does not appear in the load map.

Communicating Between Modules (Global Symbols)

Global symbols enable you to communicate between object modules. You create global symbols with the:

.GLOBL or .ENABL GBL assembler directive
Double colon (::)
Double equal sign (==)
Double equal sign and single colon (==:)

If the global symbol is defined in an object module (as a label using :: or by direct assignment using ==), other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined in some other object module. If a global symbol is used as a label in a routine, it is often called an entry point—that is, it is an entry point to that subroutine.

As the linker reads the object modules, it keeps track of all global-symbol definitions and references. It then modifies the instructions and data that reference the global symbols. The linker always displays undefined globals on the terminal after pass 1 and includes a list of undefined globals in any load maps you generate.

Example of Resolving Global References

The following table shows how the linker resolves global references when it creates the load module.

Module Name	Global Definition	Global Reference
IN1	B1 B2	A L1 C1 XXX
IN2	A B1	B2

In processing the first module, IN1, the linker finds definitions for B1 and B2, and references to A, L1, C1, and XXX. Because no definition currently exists for these references, the linker defers the resolution of these global symbols. In processing the next module, IN2, the linker finds a definition for A that resolves the previous reference, and a reference to B2 that can be immediately resolved.

When all the object modules have been processed, the linker has three unresolved global references remaining: L1, C1, and XXX. If a search of the default system library resolves XXX, the global symbols L1 and C1 remain unresolved and are, therefore, listed as undefined global symbols.

The relocatable global symbol, B1, is defined twice and is listed on the terminal as a global symbol with multiple definitions. The linker uses the first definition of such a symbol.

LINK Option Descriptions

Alphabetical (/A)

The /A option lists global symbols in program sections in alphabetical order.

Bottom-Address (/B:value[:type])

The /B option supplies the lowest address to be used by the relocatable code in the load module. The value argument is a six-digit unsigned, even octal number that defines the bottom address of the program being linked. If you do not supply a value, the linker displays:

```
?LINK-F-/B no value
```

Retype the command line, supplying an even octal value.

When you do not specify /B, the linker positions the load module so that the lowest address is location 1000_8. If the ASECT size is greater than 1000, the size of ASECT is used.

If you supply more than one /B option during the creation of a load module, the linker uses the first /B option specification. /B is invalid when you are linking to a high address (/H). The /B option is also invalid with foreground links. Foreground modules are always linked to a bottom address of 1000_8.

The bottom value must be an unsigned, even octal number. If the value is odd, the *?LINK-F-/B odd value* error message displays. Reenter the command string specifying an unsigned, even octal number as the argument to the /B option.

The Optional Type Argument

The optional type argument to the value can be DAS or INS and is used only if you also specify the /J option. When specified with /J:

- /B:value:DAS specifies the lowest address to be used by the D-space code in the load module.
- /B:value:INS specifies the lowest address to be used by the D-space code in the load module.
- /B:value:INS is the default; that is, /B:value:INS and /B:value have the same effect.

If /B is not used to specify a value for either the I-space or D-space code, 1000_8 is used as the default for that space or spaces.

/B and /H are mutually exclusive options for a particular space. However, you can use /B for one data space and /H for the other. For example, /B:value:DAS and /H:value:INS are valid to use together.

Continuation (/C or //)

The continuation option (/C or //) lets you type additional lines of command string input.

Use the /C option at the end of the current line and repeat it on subsequent command lines as often as necessary to specify all the input modules in your program. Do not enter a /C option on the last line of input.

The following command indicates that input is to be continued on the next line; the linker displays an asterisk.

```
*OUTPUT,LP:=INPUT/C  RET
*
```

An alternate way to enter additional lines of input is to use the // option on the first line. The linker continues to accept lines of input until it encounters another // option, which can be either on a line with input file specifications or on a line by itself. The advantage of using the // option instead of the /C option is that you do not have to type the // option on each continuation line. This example shows the command file that links the linker:

```
R LINK
LINK,LINK=LINK0,LNKLB1/D//
LINK1/O:1
LINK2/O:1
LINK3/O:1
LINK4/O:1
LINK5/O:1
LINK6/O:1
LINK7/O:1
LINK8/O:1
LNKEM/O:1//
BITST
GETBUF
WRIT0
WRTLRU
ZSWFIL
RET
```

You cannot use the /C option and the // option together in a link command sequence. That is, if you use // on the first line, you must use // to terminate input on the last line. If you use /C on the first line, use /C on all lines but the last.

Duplicate Global Symbol (/D)

The /D option allows you to specify library modules that you want to reside in more than one overlay segment. Type /D on the first command line. After you have typed all input command lines, the linker prompts:

```
Duplicate symbol?
```

Type the names of the global symbols in the library module that you want to be defined once in each segment that references those symbols. After each global symbol, press RETURN. If you press RETURN on a line by itself, you terminate the list of symbols.

Only global symbols defined in library modules can be duplicated. If you use the /D option and specify a global symbol that is defined outside of a library

module, the symbol definition is not duplicated and LINK displays the message *?LINK-W-Duplicate symbol SYMBOL defined in DEV:FILNAM.TYP*.

When you do not use the /D option and a global symbol defined in a library module is externally referenced (that is, the global symbol is referenced from a segment other than the one in which it is defined), the linker places the library module in the program's root segment. Therefore, if a library module is referenced by more than one global symbol, each of the global symbols in the library module that is referenced should be named in response to the /D option. Otherwise, the library module will be placed in the root segment. Also, if any of a library module's global symbols are referenced from the root, the library module will be placed in the root even if you have named the global symbols in response to the /D option. In each of these cases, when a library module that LINK places in the root contains global symbols declared with /D, LINK displays the following message and the global symbol is not duplicated:

```
?LINK-W-Duplicate symbol SYMBOL is forced to the root
```

Special Programming Considerations for the /D Option

Even when a library module you duplicate is not referenced from the root, any global section within that module that is referenced from more than one segment is always placed in the root. If local sections within the same library module have no need to communicate with each other, define the global section with the CON attribute. This causes the linker to place a separate copy of the global section in the root for each copy of the library module's local sections placed in overlays. Although the global section resides in the root while the local sections reside in overlays, each copy of the library module retains its identity as a separate copy of the module. Since each copy of the global section is bound to its own local section in an overlay, this ensures that references between the local and global sections will be bound to the correct definitions.

However, when a library module that you want to duplicate is placed in overlay segments that exchange information, another consideration exists. If the library module contains a section of global data to be referenced by local sections within the module, but the global section does not reference any local section within the module, you should move a copy of the global section to the root. To move this section to the root, define the section with a unique name and give the section the GBL and OVR attributes. When this section is placed in the root, the local sections from the duplicated library module that reside in the overlay segments can reference the global section in the root. Since the global section has been given the OVR attribute rather than CON, the local sections can pass information to specific locations in the global section, and the local sections can access the same locations to send and receive data.

Figure 15–3 illustrates a duplicated library module whose global data section has been forced to the root with the CON attribute. The arrows show each local section accessing information from its copy of the global section within the root. Notice, however, that the local sections (which are identical) cannot exchange data because their references are bound to different locations. Figure 15–4 illustrates the same duplicated library module, this time with the global data

section forced to the root with the OVR attribute. Notice that the two local sections can now reference the same location in the global section to exchange information.

Figure 15–3: Global Data Section with CON Attribute

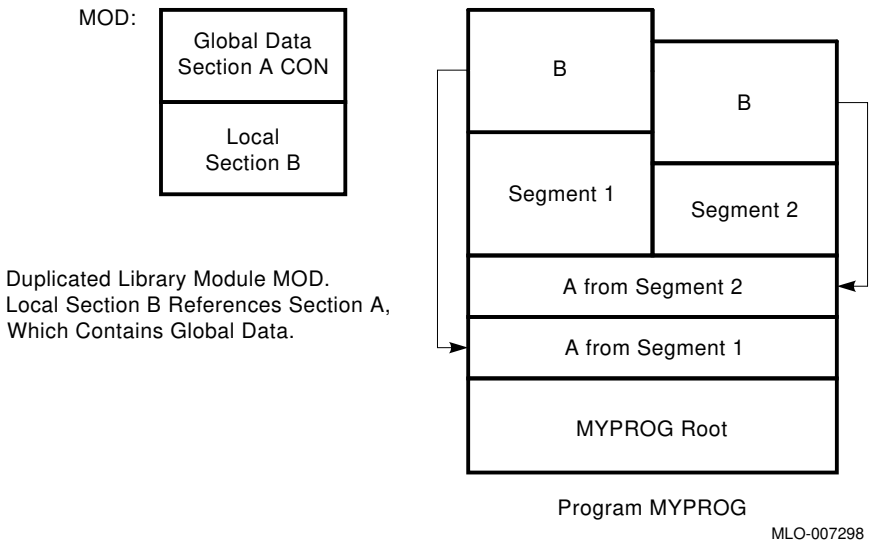
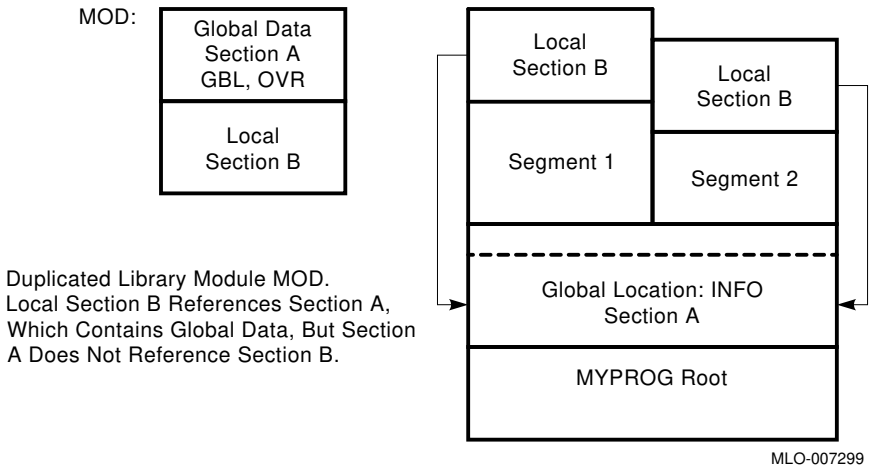


Figure 15–4: Global Data Section with OVR Attribute



Extend Program Section (/E:value[:type])

The /E:value option allows you to extend a program section in the root to a specific value. Type the /E:value option at the end of the first command line.

The Optional Type Argument

The optional type argument to the value can be DAS or INS and is used only if you also specify the /J option. When specified with /J:

- /E:value:DAS specifies the minimum size to allocate to a D-space PSECT that you specify.
- /E:value:INS specifies the minimum size to allocate to an I-space PSECT that you specify.
- /E:value:INS is the default; that is, /E:value:INS and /E:value have the same effect.

When you have entered the complete LINK command, RT–11 prompts you for the name of the program section you need to extend:

- If you do not also use the /J option, the prompt is:

  ```
  Extend section?
  ```

  Respond with the name of the program section to be extended and press RETURN. The resultant program section size is equal to or greater than the value you specify, depending on the space the object code requires. The value you specify must be an even byte value. Note that you can extend only one section.

  The following example extends section CODE to 100_8 bytes.

  ```
  *X,TT:=LK001/E:100 RET
  Extend section? CODE RET
  ```

- If you use the /J option, the prompt is either one or both of the following, depending on whether one or both types of /E are specified. If both types are specified, the prompts are issued in the following order:

  ```
  Extend instruction section?
  Extend data section?
  ```

Respond with the appropriate program section name(s), and terminate your response with RETURN. The sections specified in answer to these prompts are verified to be I-space or D-space sections, as appropriate. If not, an error message is generated.

Default FORTRAN Library (/F)

By indicating the /F option in the command line, you can link the FORTRAN library (FORLIB.OBJ on the system device SY:) with the other object modules you specify. You do not need to specify FORLIB explicitly. For example:

```
*FILE,LP:=AB/F RET
```

The object module AB.OBJ from DK and the required routines from the FORTRAN library SY:FORLIB.OBJ are linked together to form a load module called FILE.SAV.

The linker automatically searches a default system library, SY:SYSLIB.OBJ. The library normally includes the modules that compose FORLIB. The /F option is provided only for compatibility with other versions of RT–11. You should not have to use /F.

See the *RT–11 Programmer's Reference Manual* and the *RT–11 Installation Guide* for details on combining SYSLIB and FORLIB library files.

Directory Buffer Size (/G)

When you are using modules for your program that are from a multiple-definition library, LINK has to store that library's directory in an internal buffer. Occasionally, this buffer area is too small to contain an entire directory, in which case LINK is unable to process those modules. The /G option instructs LINK to adjust the size of its directory buffer to accommodate the largest directory size of the multiple-definition libraries you are using.

You should use /G only when required because it slows down linking time. Use it only after an attempt to link your program failed because the buffer was too small. When a link failure of this sort occurs, LINK displays the message *?LINK-F-Library EPT too big, increase buffer with /G*.

Highest Address (/H:value[:type])

The /H:value option allows you to specify the top (highest) address to be used by the relocatable code in the load module. The value argument specifies an unsigned, even octal number. If you do not specify a value, the linker displays:

```
?LINK-F-/H no value
```

Retype the command, supplying an even octal number to be used as the value.

If you specify an odd value, the linker responds with:

```
?LINK-F-/H odd value
```

Retype the command, supplying an even octal number.

If the value is not large enough to accommodate the relocatable code, the linker displays:

```
?LINK-F-/H value too low
```

Relink the program with a larger value.

The Optional Type Argument

The optional type argument to the value can be DAS or INS and is used only if you also specify the /J option. When specified with /J:

- /H:value:DAS specifies the highest address to be used by the D-space code in the load module. The *value* must be even.
- /H:value:INS specifies the highest address to be used by the I-space code in the load module. The *value* must be even.

- /H:value:INS is the default; that is, /H:value:INS and /H:value have the same effect.

The /H option cannot be used with the /R, /Y, or /B options.

The /B and /H options are mutually exclusive for a particular space. However, you can use /B for one data space and /H for the other. For example, /B:value:DAS and /H:value:INS are valid to use together.

The /Y and /H options are also mutually exclusive for a particular space. However, you can use /Y for one data space and /H for the other. For example, /Y:value:DAS and /H:value:INS are valid to use together.

NOTE

Be careful when you use the /H option. Most RT–11 programs use the free memory above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, and so on. The size of this area differs according to the memory configuration. Programs linked to a specific high address might not run in a system with less physical memory because there is less free memory.

Include (/I)

The /I option lets you take global symbols from any library and include them in the linking process even when they are not needed to resolve globals. This provides a method for forcing modules that are not called by other modules to be loaded from the library. All modules that you specify with /I go into the root. When you specify the /I option, the linker displays:

```
Library search?
```

Reply with the list of global symbols to be included in the load module. Press RETURN to enter each symbol in the list, and press RETURN alone to terminate the list of symbols.

The following example includes the global $SHORT in the load module:

```
*SCCA=RK1:SCCA/I RET
Library search? $SHORT RET
Library search? RET
```

Separated Instruction and Data Space (/J)

The /J option causes LINK to generate an extended SAV image file which separates I- and D-space. Specify /J on the first line of the LINK command to produce a separated I- and D-space program. The following options are modified based on the presence of this option: /B, /E, /H, /M, /Q, /T, /U, /X, /Y, and /Z. See the individual option descriptions for more information. The /R and /J options are incompatible and generate an error message when used together.

Memory Size (/K:value)

The /K:value option lets you insert a value into word 56 of block 0 of the image file. The value specifies the number of 1K words of memory required by the program; the value is an integer in the range $2–28_{10}$. You cannot use the /K option with the /R option.

The /K:value option is provided for compatibility with the RSTS/E operating system. RT–11 ignores information provided by the /K:value option, although word 56 in block 0 of the image file is modified.

LDA Format (/L)

The /L option produces an output file in LDA format instead of memory-image format. The LDA format file can be output to any device including those that are not block-replaceable. It is useful for files that are to be loaded with the absolute loader. The default file type LDA is assigned when you use the /L option. You cannot use the /L option with the low-memory overlay option (/O), the foreground link option (/R), or the extended-memory overlay option (/V).

The following example links files IN and IN2 on device DK and outputs an LDA format file, OUT.LDA, to the diskette and a load map to the line printer.

```
*DY:OUT,LP:=IN,IN2/L RET
```

Modify Stack Address (/M[:value])

The stack address, location 42, that contains the initial value for the stack pointer. The /M option lets you specify the stack address. If you use the /R:stacksize option (foreground link) with /M, LINK ignores the value on /R:stacksize. The value argument is an even, unsigned, six-digit octal number that defines the stack address.

After all input lines have been typed, the linker displays the following message if you have not specified a value:

```
Stack symbol?
```

In this case, specify the global symbol whose value is the stack address and press RETURN. You must not specify a number. If you specify a nonexistent symbol, an error message displays and the stack address is set to the system default (1000 for SAV files) or to the bottom address if you used /B. If the program's absolute section extends beyond location 1000, the default stack space starts after the largest .ASECT allocation of memory.

Direct assignment (with .ASECT) of the stack address within the program takes precedence over assignment with the /M option. The statements to do this in a MACRO program are as follows:

```
.ASECT
.=42
.WORD  INITSP              ;INITIAL STACK SYMBOL VALUE
.PSECT                     ;RETURN TO PREVIOUS SECTION
```

The following example modifies the stack address.

```
*OUTPUT=INPUT/M RET
Stack symbol? BEG RET
```

If you specify /M together with /J but do not specify a value for /M, the specified stack symbol is verified to be in D-space. If it is not, an error is generated.

Cross Reference (/N)

The /N option includes in the load map a cross-reference of all global symbols defined during the linking process. The global symbols are listed alphabetically. Each global symbol is followed by the names of the modules (also listed alphabetically) in which the symbol is defined or referenced. A pound sign (#) next to the module name indicates that the symbol is defined in that module. A plus sign (+) indicates that the module is from a library. The cross-reference section, if requested, begins on a new page at the end of the load map. See Figure 15–2 and the Example of Resolving Global References section for an illustration of how a global cross-reference listing is done.

When you request a global symbol cross-reference listing with the /N option, LINK generates the temporary file DK:CREF.TMP.

If DK is write-locked or if it contains insufficient free space for the temporary file, you can designate another device for the file. To designate another device for the temporary file, assign the logical name CF to the device by using the following command:

```
.ASSIGN dev: CF RET
```

If you have assigned CF to a physical device for MACRO cross-reference listing temporary file CREF.TMP, that device will also serve as the default device for the LINK global symbol cross-reference temporary file.

Low-Memory Overlay (/O:value)

The /O option segments the load module so that the entire program is not memory resident at one time. This lets you execute programs that are larger than the available memory.

The *value* argument is an unsigned octal number (up to five digits) specifying the overlay region to which the module is assigned. The /O option must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line; that is, reserved for the root segment. You must use /C or // for continuation.

You specify coresident overlay routines (a group of subroutines that occupy the overlay region and segment at the same time) as follows:

```
*OBJA,OBJB,OBJC/O:1/C RET
*OBJD,OBJE/O:2/C RET
  .
  .
  .
```

All modules that the linker encounters until the next /O option will be coresident overlay routines. If you specify, at a later time, the /O option with the same value you used previously (same overlay region), then the linker opens up the corresponding overlay area for a new group of subroutines. This group occupies the same locations in memory as the first group, but it is never needed at the same time as the previous group.

The following commands to the linker make R and S occupy the same memory as T (but at different times):

```
*MAIN,LP:=ROOT/C  RET
*R,S/O:1/C  RET
*T/O:1  RET
```

The following example establishes two overlay regions.

```
*OUTPUT,LP:=INPUT//  RET
*OBJA/O:1  RET
*OBJB/O:1  RET
*OBJC/O:2  RET
*OBJD/O:2  RET
*//  RET
```

You must specify overlays in ascending order by region number. For example:

```
*A=A/C  RET
*B/O:1/C  RET
*C/O:1/C  RET
*D/O:1/C  RET
*G/O:2  RET
```

The following overlay specification is invalid since the overlay regions are not given in ascending numerical order. An error message displays in each case, and the overlay option immediately preceding the message is ignored.

```
*X=LIBR0//  RET
*LIBR1/O:1  RET
*LIBR2/O:0  RET
?LINK-W-/O or /V option error, re-enter line
*
```

In the above example, the overlay line immediately preceding the error message is ignored, and should be re-entered with an overlay region number greater than or equal to one.

Library List Size (/P:value)

The /P:value option lets you change the amount of space allocated for the library routine list. Normally, the default value allows enough space for your needs. It reserves space for approximately 170 unique library routines, which is the equivalent of specifying /P:$170._{10}$ or /P:252_8. See the *RT–11 Installation Guide* for details on customizing this default number for the library routine list.

The error message *?LINK-F-Library list overflow, increase size with /P* indicates that you need to allocate more space for the library routine list. You must relink

the program that makes use of the library routines. Use the /P:value option and supply a value that is greater than 170_{10}.

You can use the /P:value option to correct for symbol table overflow. Specify a value that is less than 170. This reduces the space used by the library routine list and increases the space allocated for the symbol table. If the value you choose is too small, the *?LINK-F-Library list overflow, increase size with /P* message displays.

In the following command, the amount of space for the library routine list is increased to 300_{10}.

```
*SCCA=RK1:SCCA/P:300.  RET
```

Absolute Base Address (/Q)

The /Q option lets you specify the absolute base addresses of up to eight PSECTs in your program. This option is particularly handy if you are preparing your program sections in absolute loading format for placement in ROM storage.

When you use this option in the first command line, the linker prompts you for the PSECT names and load addresses. The PSECT name must be six characters or less, and the load address must be an even octal number. Terminate each line with RETURN. If you only press RETURN in response to any of the prompts, LINK ceases prompting.

When the /Q option is used with the /J option, /Q can refer to I-space or D-space PSECTs intermixed in any fashion.

If you use /E, /Y, or /U with /Q, LINK processes those options before it processes /Q.

When you use the /Q option, observe the following restrictions:

- Enter only even addresses. If you enter an odd address, no address, or invalid characters, LINK displays an error message and then prompts you again for the PSECT and load address.
- /Q is invalid with /H or /R. These options are mutually exclusive.
- LINK moves your PSECTs up to the specified address; moving down might destroy code. If your address requires code to be moved down, LINK displays an error message, ignores the PSECT for which you have specified a load address, and continues.

The following example specifies the load addresses for three PSECTs.

```
*FILE,TT:=FILE,FILE1/Q/L  RET
Load Section:Address?  PSECT1:1000  RET
Load Section:Address?  PSECT3:4000  RET
Load Section:Address?  PSECT2:2500  RET
Load Section:Address?  RET
```

REL Format (/R[:stacksize])

The /R[:stacksize] option produces an output file in REL format for use as a foreground job with a multi-job monitor. You cannot use REL files under the SB monitor. The /R option assigns the default file type REL to the output file. The optional stacksize argument specifies the amount of stack space to allocate for the foreground job; it must be an even octal number. The default value is 128_{10} bytes of stack space. If you also use the /M option, the value or global symbol associated with it overrides the /R value.

The following command links files FILEI.OBJ and NEXT.OBJ and stores the output on DY1: as FILEO.REL. It also prints a load map on the printer:

```
*DY1:FILEO,LP:=FILEI,NEXT/R:200 RET
```

You cannot use the /B, /H, J/, or /L option with /R since a foreground REL job has a temporary bottom address of 1000 and is always relocated by FRUN. An error message displays if you attempt this. The /K option is also invalid with /R.

Symbol Table (/S)

The /S option instructs the linker to allow the largest possible memory area for its symbol table at the expense of input and output buffer space. Because this makes the linking process slower, you should use the /S option only if an attempt to link a program failed because of symbol table overflow. When you use /S, do not specify a symbol table file or a map in the command string.

Transfer Address (/T[:value])

The transfer address is the address at which a program starts when you initiate execution with an R, RUN, SRUN (GET, START), or FRUN command. It displays on the last line of the load map. The /T option lets you specify the start address of the load module. The value argument is a six-digit, unsigned, even octal number that defines the transfer address.

If you have not also specified the /J option and you do not specify the value, the following message displays:

```
Transfer symbol?
```

In this case, specify the global symbol whose value is the transfer address of the load module. Terminate your response by pressing RETURN. You cannot specify a number in answer to this message. If you specify a nonexistent symbol, an error message displays and the transfer address is set to 1 so that the program traps immediately if you attempt to execute it. If the transfer address you specify is odd, the program does not start after loading and control returns to the monitor.

If you specify /T with /J and do not specify a value for /T, the specified transfer symbol is verified to be in I-space. If it is not, an error message is displayed.

Direct assignment (.ASECT) of the transfer address within the program takes precedence over assignment with the /T option. The transfer address assigned with a /T option has precedence over that assigned with an .END assembly directive. To assign the transfer address within a MACRO program, use statements similar to these:

```
          .ASECT
          .=40
          .WORD       START1      ;SYMBOL VALUE FOR TRANSFER ADDRESS
          .PSECT                  ;RETURN TO PREVIOUS SECTION
START1:   .
          .
          .
or
START2:   .                       ;SECONDARY STARTING ADDRESS
          .
          .
          .END         START2
```

The following example links the files LIBR0.OBJ and ODT.OBJ together and starts execution at ODT's transfer address.

```
*LBRODT,LBRODT=LIBR0,ODT/T/W//  RET
*LIBR1/O:1  RET
*LIBR2/O:1  RET
*LIBR3/O:1  RET
*LIBR4/O:1  RET
*LIBR5/O:1  RET
*LIBR6/O:1  RET
*LBREM/O:1//  RET
Transfer symbol? O.ODT  RET
*
```

Round Up (/U:value[:type])

The /U:value option rounds up the section you name in the root so that the size of the root segment is a whole number multiple of the value you specify. The value argument must be a power of 2.

The Optional Type Argument

The optional type argument to the /U value can be DAS or INS and is used only if you also specify the /J option. When specified with /J:

- /U:value:DAS specifies the size boundary for the D-space root. This size must be an integer multiple of *value*; that is, *value* must be a power of 2. The size of the specified D-space PSECT is rounded up the minimum amount necessary to accomplish this.
- /U:value:INS specifies the size boundary for the I-space root. This size must be an integer multiple of *value*; that is, *value* must be a power of 2. The size of the specified I-space PSECT is rounded up the minimum amount necessary to accomplish this.
- /U:value:INS is the default; that is, /U:value:INS and /U:value have the same effect.

When you specify the /U:value option:

- If you do not also specify the /J option, the linker prompts:

  ```
  Round section?
  ```

Reply with the name of the program section to be rounded and press RETURN. The program section must be in the root segment. Note that you can round only one program section.

The following example rounds up section CHAR.

```
*LK007,TT:=LK007/U:200 RET
Round section? CHAR RET
```

If the program section you specify cannot be found, the linker displays *?LINK-W-Round section not found AAAAAA* and the linking process continues with no rounding.

- If you also specify the /J option, the prompt is either one or both of the following, depending on whether one or both types of /U are specified. If both types are specified, the prompts are issued in the following order:

```
Round instruction section?
Round data section?
```

Respond with the appropriate program section name(s), and terminate your response with RETURN. The sections specified in answer to these prompts are verified to be I-space or D-space sections, as appropriate. If not, an error message is generated.

Extended-Memory Overlay (/V:value-a[:value-b])

Use the /V option to create an extended-memory overlay structure for your program. The /V option describes your program's structure in terms of virtual overlay regions (areas of virtual address space) and partitions (areas of physical address space). The *value-a* argument specifies a virtual overlay region, and *value-b* specifies a partition. As you specify successive extended-memory overlay segments in the command string, make sure that value-a and value-b in the /V:value-a[:value-b] notation are in ascending order.

If you use /V on the first command line with no arguments, you enable special .SETTOP features provided by a mapped monitor and special .LIMIT features. When used on the first line of the command string, this option allows virtual or privileged foreground or background jobs to map a work area in extended memory with the .SETTOP programmed request. Thus, your program does not need an extended-memory overlay structure to make use of the mapped-monitor .SETTOP features. See the *RT–11 System Macro Library Manual* and the *RT–11 Volume and File Formats Manual* for more details on these features and extended memory.

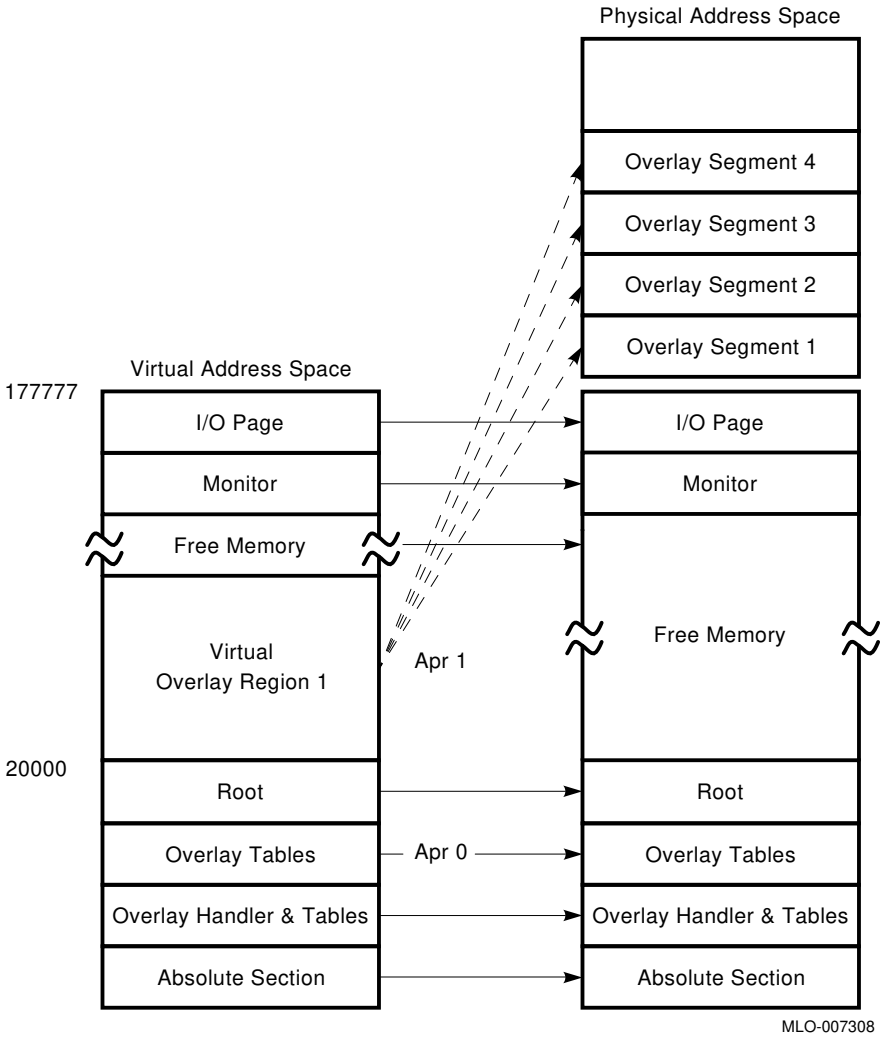
The following examples show how to use the /V:value-a[:value-b] option.

1. In this example, program PROG has four segments to be mapped to extended memory. The four segments are named SEG1, SEG2, SEG3, and SEG4:

```
.R LINK  RET
*PROG=PROG//  RET
*SEG1/V:1  RET
*SEG2/V:1  RET
*SEG3/V:1  RET
*SEG4/V:1//  RET
```

   These segments map into extended memory exactly as shown in Figure 15–5.

Figure 15–5: Virtual and Physical Address Space with One Virtual Region



Notice how each segment fits into its own partition in extended memory. Because each segment fits into its own partition, no storage volume access is necessary to change (or swap) segments once they have been read in.

NOTE

The /V:value-a[:value-b] option works differently from the /O:value option. If /O:value were used in the previous example, the four segments would share the same physical locations, obviously requiring storage volume I/O as each segment is called. With /V:value-a[:value-b], each segment from the previous example occupies a unique area in extended memory, and no mass storage I/O is necessary after each segment is called.

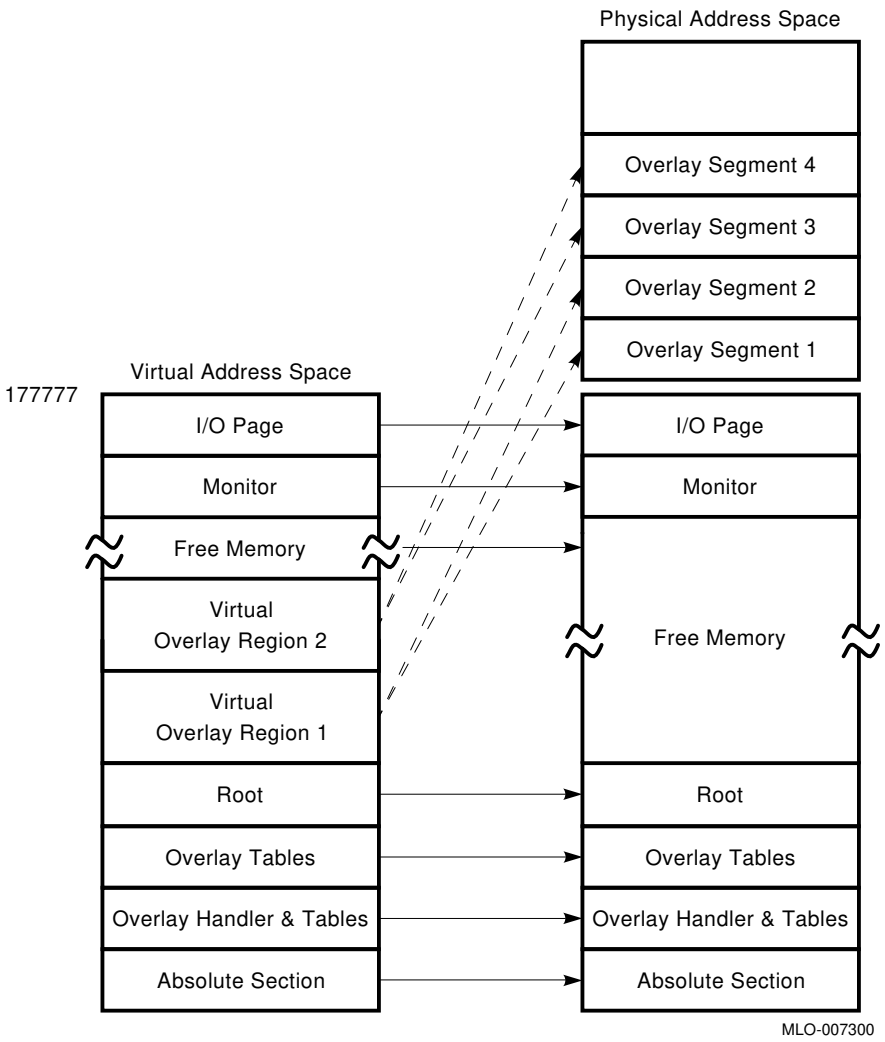
2. This example places the same four segments as described in the previous example into virtual overlay regions 1 and 2. Although the program in this example uses two virtual overlay regions at run time, the segments will reside in memory the same as the segments shown in Figure 15–5.

   The virtual address space, however, will be different for this example. SEG1 and SEG2 use APR 1 (20000 to 37777), while SEG3 and SEG4 use APR 2 (40000 to 57777):

```
.R LINK RET
*PROG=PROG// RET
*SEG1/V:1 RET
*SEG2/V:1 RET
*SEG3/V:2 RET
*SEG4/V:2// RET
```

Figure 15–6 shows how this example is mapped to memory.

Figure 15–6: Virtual and Physical Address Space with Two Overlay Regions



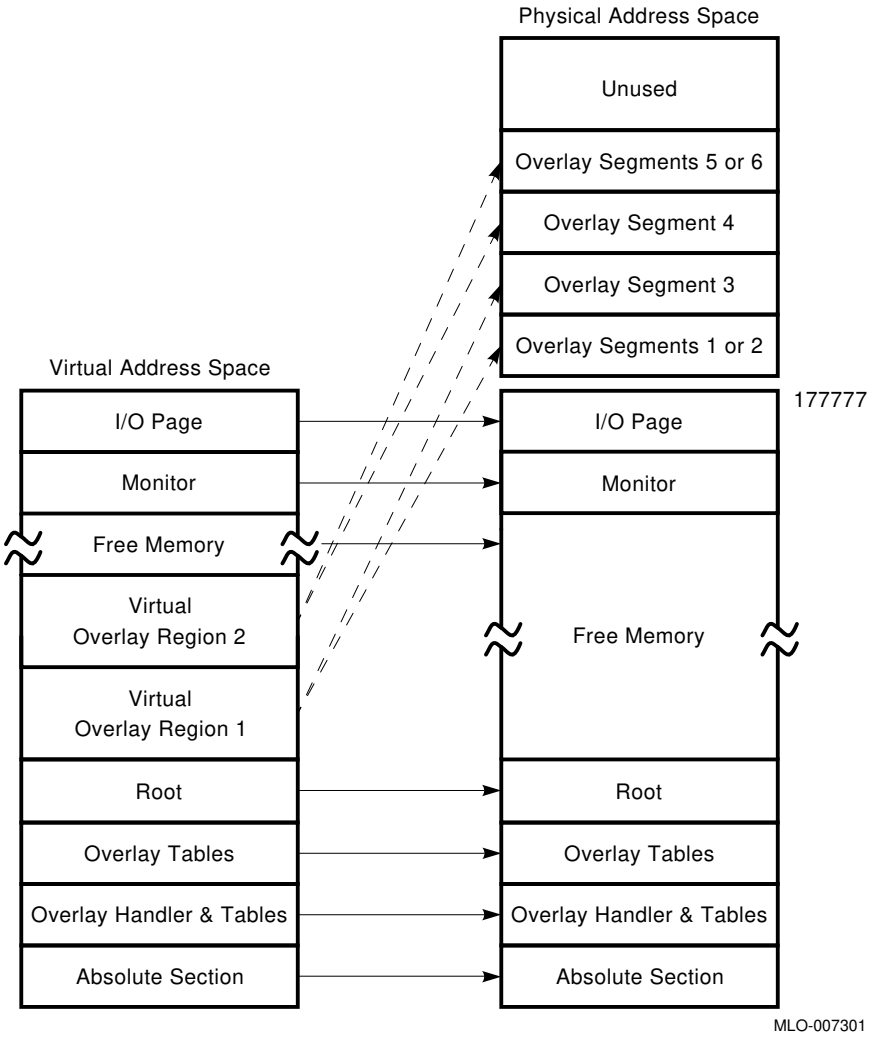
The value-b argument in /V:value-a[:value-b] specifies the partition in extended memory for the overlay segment. If you use value-b, segments can share the same partition in extended memory. That is, a segment, when called by your program, can be read in from auxiliary storage, thus overlaying the segment that currently occupies the same partition. When segments share partitions, the program requires auxiliary storage for I/O during run time, as does a program with low-memory overlays.

LINK Option Descriptions

LINK makes each partition the size of the largest segment it must accommodate. The following example generates the overlay structure shown in Figure 15–7:

```
.R LINK  RET
*PROG=PROG//  RET
*SEG1/V:1:1  RET
*SEG2/V:1:1  RET
*SEG3/V:2  RET
*SEG4/V:2  RET
*SEG5/V:2:1  RET
*SEG6/V:2:1//  RET
```

Figure 15–7: Extended-Memory Partitions that Contain Sharing Segments



Notice that there are four segments specified for virtual overlay region 2, and that two segments share partition 1. Value-b in /V:value-a:value-b groups segments in a region. The only reason to use value-b is to create a partition that contains

two or more segments. As shown in the previous example, value-b is specified in ascending order within each virtual overlay region. This means you can renumber value-b from 1 for each virtual overlay region.

If you specify four segments for the same virtual overlay region, as in Example 1 below, the result is the same as if you specified Example 2. Because two segments are not specified to share the same partition, the partition order is as Example 2 shows.

Example 1

```
*SEG1/V:1  RET
*SEG2/V:1  RET
*SEG3/V:1  RET
*SEG4/V:1  RET
```

Example 2

```
*SEG1/V:1:1  RET
*SEG2/V:1:2  RET
*SEG3/V:1:3  RET
*SEG4/V:1:4  RET
```

Map Width (/W)

The /W option directs the linker to produce a wide load map listing. If you do not specify the /W option, the listing is wide enough for three global value columns (normal for paper with 80-character columns). If you use the /W command, the listing is six columns wide, which is suitable for a 132-column page.

Bitmap Inhibit (/X)

The /X option instructs the linker not to output the bitmap if code lies in locations 360 to 377 inclusive. This option is provided for compatibility with the RSTS operating system. The bitmap is stored in locations 360–377 in block 0 of the load module, and the linker normally stores the program memory usage bits in these eight words. Each bit specifies one 256-word block of memory. This information is required by the R, RUN, and GET commands when loading the program; therefore, use care when you use this option.

The /X option causes both the I- and D-space bitmaps to be suppressed, while the absence of /X causes both I- and D-space bitmaps to be generated. One cannot be generated and the other suppressed.

Boundary (/Y[:value[:type]])

The /Y[:value] option starts a specific program section in the root on a particular address boundary. Do not use this option with /H. The linker generates a whole number multiple of *value*, the value you specify, for the starting address of the program section. The *value* argument must be a power of 2. The linker extends the size of the previous program section to accommodate the new starting address.

When You Do Not Specify a Value

The *value* argument for the /Y option is optional. If you do not specify the *value* argument, LINK prompts for up to eight separate PSECT boundary addresses. The prompt is:

```
Boundary section?
```

Respond to the prompt with the name of the program section whose starting address you are modifying. Use the form:

```
symbol[:m]
```

where *symbol* is the PSECT name, and *m* is the address boundary you assign the PSECT. Terminate your response by pressing RETURN.

If *m* is not specified, any value that was entered at the /Y[:value] option is used, and prompting stops. If a value is not specified at the command line or the prompt, the default value 1000_8 is used as the boundary address.

The Optional Type Argument

The optional type argument to the value can be DAS or INS and is used only if you also specify the /J option. When specified with /J:

- /Y:value:DAS specifies a particular address boundary at which a specified D-space PSECT in the root begins.
- /Y:value:INS specifies a particular address boundary at which a specified I-space PSECT in the root begins.
- /Y:value:INS is the default; that is, /Y:value:INS has the same effect as /Y:value.

/Y and /H are mutually exclusive options for a particular space. However, you can use /Y for one data space and /H for the other. For example, /Y:value:DAS and /H:value:INS are valid to use together.

If Program Section Cannot Be Found

If the program section you specify cannot be found, the linker displays *?LINK-W-Boundary section not found*, and the linking process continues.

The RT–11 monitors have internal two-block overlays. The first overlay segment, OVLY0, must start on a disk-block boundary:

```
*RT11SJ.SYS=BTSJ,RMSJ,KMSJ,TBSJ/Y:1000 RET
Boundary section? OVLY0 RET
```

Boundary Prompts

When you have entered the complete LINK command, RT–11 prompts you for the name of the section whose starting address you need to modify.

- If you do not also use the /J option, the prompt is:

  ```
  Boundary section?
  ```

- If you use the /J option, the prompt is either one or both of the following, depending on whether one or both types of /Y are specified. If both types are specified, the prompts are issued in the following order:

```
Instruction boundary section?
Data boundary section?
```

Respond with the appropriate value and/or program section name(s), and terminate your response by pressing RETURN. The sections specified in answer to these prompts are verified to be I-space or D-space sections, as appropriate. If not, an error message is generated.

If you do not want to specify a value, respond with only the appropriate program section name. If you want to specify a value, respond in the following format:

value[:type]

where:

value	specifies the address boundary you assign that PSECT.
type	specifies the abbreviation for the PSECT name, the name of the section whose starting address you need to modify. DAS is for the data section and INS is for the instruction section.

Not specifying the value parameter causes LINK to prompt for up to eight separate PSECT boundary addresses. You terminate the prompt sequence by pressing RETURN with no specified value.

If you do not specify a value, any value that was entered at the /Y:value option is used, and prompting stops. If a value is not specified at the command line or the prompt, the default value 1000_8 is used as the boundary address.

Zero (/Z:value[:type])

The /Z:value option fills unused locations in the load module and places a specific value in these locations. This option can be useful in eliminating random results that occur when the program references uninitialized memory by mistake. RT–11 automatically zeroes unused locations. Use the /Z:value option only when you want to store a value other than zero in unused locations. You cannot use the R, RUN, FRUN, or GET commands to load into memory a load image block of fill characters.

The Optional Type Argument

The optional type argument to the value can be DAS or INS and is used only if you also specify the /J option. When specified with /J:

- /Z:value:DAS initializes all unused D-space locations in the load module with *value*.
- /Z:value:INS initializes all unused I-space locations in the load module with *value*.
- /Z:value:INS is the default, which means that /Z:value:INS has the same effect as /Z:value.

LINK Option Summary

Table 15–6 lists the options associated with the linker. The second column, titled *Command Line*, lists on which line in the command string the option in column one can be placed. If you continue the input on more than one line, you must place the options on the line indicated, though you can position them anywhere on that line. The LINK Option Descriptions section describes each option in more detail.

Table 15–6: Linker Options

Option	Command Line	Function
/A	First	Lists global symbols in program sections in alphabetical order.
/B:value[:type]	First	Changes the bottom address of a program to value (invalid with /H and /R).
/C	Any but last	Continues input specification on another command line. (You can also use /C with /V and with /O; do not use /C with the // option.)
/D	First	Allows the global symbol you specify to be defined once in each segment that references that symbol. These symbols must be defined in library modules.
/E:value[:type]	First	Extends a particular program section in the root to a specific value.
/F	First	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ; this option is provided only for compatibility with previous versions of RT–11.
/G	First	Adjusts the size of the linker's library directory buffer to accommodate the largest multiple-definition library directory.
/H:value[:type]	First	Specifies the top (highest) address to be used by the relocatable code in the load module. Invalid with /B, /R, /Y and /Q.
/I	First	Extracts the global symbols you specify (and their associated object modules) from the library and links them into the load module.
/J	First	Causes LINK to generate an extended SAV image file which separates I- and D-space.
/K:value	First	Inserts the value you specify (the valid range for the value is from 2 to 28.) into word 56 of block 0 of the image file. This option allows you to limit the amount of memory allocated by a .SETTOP request to n K $words_{10}$. Invalid with /R.
/L	First	Produces a formatted binary output file (invalid for overlaid programs and for foreground links).

Table 15–6 (Cont.): Linker Options

Option	Command Line	Function
/M[:value]	First	Causes the linker to prompt you for a global symbol that specifies the stack address or that sets the stack address to the specified value. Do not use with /R.
/N	First	Produces a cross-reference in the load map of all global symbols defined during the linking process.
/O:value	Any but first	Indicates that the program is an overlay structure; the value specifies the overlay region to which the module is assigned. Invalid with /L.
/P:value	First	Changes the default amount of space the linker uses for a library routines list.
/Q	First	Lets you specify the base addresses of up to eight root program sections. Invalid with /H or /R.
/R[:stacksize]	First	Produces output in relocatable format and optionally indicates stack size for a foreground job. Invalid with /B, /H, /K, and /L.
/S	First	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this option only when a particular link stream causes a symbol table overflow.)
/T[:value]	First	Causes the linker to prompt you for a global symbol that specifies the transfer address or that sets the transfer address to the specified value.
/U:value[:type]	First	Rounds up the root program section you specify so that the size of the root segment is a whole number multiple of the value you supply (n must be a power of 2).
/V	First	Enables special .SETTOP and .LIMIT features provided by the XM monitor. Invalid with /L.
/V:value-a[:value-b]	Any but last	Indicates that an extended memory overlay segment is to be mapped in virtual region value-a, and optionally in partition value-b.
/W	First	Directs the linker to produce a wide load map listing.
/X	First	Does not output the bitmap if the code is placed over the bitmap (location 360-377). This option is provided only for compatibility with the RSTS operating system.

Table 15–6 (Cont.): Linker Options

Option	Command Line	Function
/Y[:value[:type]]	First	Starts a specific program section in the root on a particular address boundary. Invalid with /H.
/Z:value[:type]	First	Sets unused locations in the load module to the specified value.
//	First and last	Allows you to specify command string input on additional lines. Do not use this option with /C.

DCL Equivalents of LINK Utility Operations

Table 15–7 lists all the LINK utility CSI operations.

The first part of the table lists that part of the CSI LINK command syntax that is equivalent to a DCL LINK option. The rest of the table alphabetically lists all the CSI LINK options, both those with and those without DCL equivalents. An asterisk indicates a CSI option that has no DCL equivalent.

Note that the CSI options /F, /G, /O:n, /P:n, and /Q have no DCL equivalents, while the DCL options /DEBUG[:filespec], /LIBRARY:filespec, /LINKLIBRARY:filespec, and /RUN have no CSI equivalents. See the *RT–11 Commands Manual* for a description of all the LINK DCL options.

Table 15–7: DCL Equivalents of LINK Utility Operations

CSI Command/Option	DCL Option
1st output filespec	/EXECUTE[:filespec]
no 1st output filespec	/NOEXECUTE[:filespec]
filespec[size]= (1st output filespec)	/ALLOCATE:size
2nd output filespec	/MAP[:filespec]
3rd output filespec	/SYMBOLTABLE[:filespec]
/A	/ALPHABETIZE
/B:value[:type]	/BOTTOM:value[:type]
/C	/PROMPT
/D	/DUPLICATE
/E:value[:type]	/EXTEND:value[:type]
/F	*
/G	*
/H:value[:type]	/TOP:value[:type]
/I	/INCLUDE
/J	/IDSPACE
/K:value	/LIMIT:value
/L	/LDA
/M[:value]	/STACK[:value]
/N	/GLOBAL
/O:value	*
/P:value	*

Table 15–7 (Cont.): DCL Equivalents of LINK Utility Operations

CSI Command/Option	DCL Option
/Q	*
/R[:stacksize]	/FOREGROUND[:stacksize]
/S	/SLOWLY
/T[:value]	/TRANSFER[:value]
/U:value[:type]	/ROUND
/V:value-a[:value-b]	/XM
/W	/WIDE
/X	/NOBITMAP
/Y:value[:type]	/BOUNDARY:value[:type]
/Z:value[:type]	/FILL:value[:type]
//	/PROMPT

Appendix A
The Linker Overlays

The RT–11 linker can overlay:

Low memory
Extended (virtual) memory
Separated I (instruction) and D (data) space

This appendix includes the code for the overlay handlers, describes how programs use these handlers, and shows how LINK uses overlays. See Chapter 15 for a description of the LINK options enabling the overlay handlers.

Creating an Overlay Structure

The ability of RT–11 to handle overlays gives you virtually unlimited memory space for an assembly language or FORTRAN program. A program using overlays can be much larger than would normally fit in the available memory space, since portions of the program reside on a storage device such as a disk. To utilize this capability, you must define an overlay structure for your program.

The overlay handlers, enabling the overlay functions, are included in the SYSLIB library and used by the linker:

- The OHANDL overlay handler maps low-memory overlays, that is, overlays that reside in low memory. Prior to Version 4, RT–11 permitted overlays to be placed only in low memory.
- The VHANDL overlay handler maps extended-memory overlays, that is, overlays that reside in extended memory. If you run your program on a system that has an extended-memory configuration and a mapped monitor, you can have extended-memory overlays.

  The Low-Memory Overlays section describes low-memory overlays in general and shows how to define a low-memory overlay structure for your program. The Extended-Memory Overlays section deals specifically with extended-memory overlays, and shows how to define an overlay structure that has either extended-memory overlays only or both extended-memory and low-memory overlays. Read the Low-Memory Overlays section before reading the Extended-Memory Overlays section, because much of the information contained in the first section applies to the second section.
- The XHANDL overlay handler, uses less low memory than VHANDL and maps a single virtual overlay segment.
- The ZHANDL and SHANDL handlers map separated I (instruction) and D (data) space.

SHANDL contains the source code for all the overlay handlers. Conditionals in SHANDL build the different variants of the overlay handler.

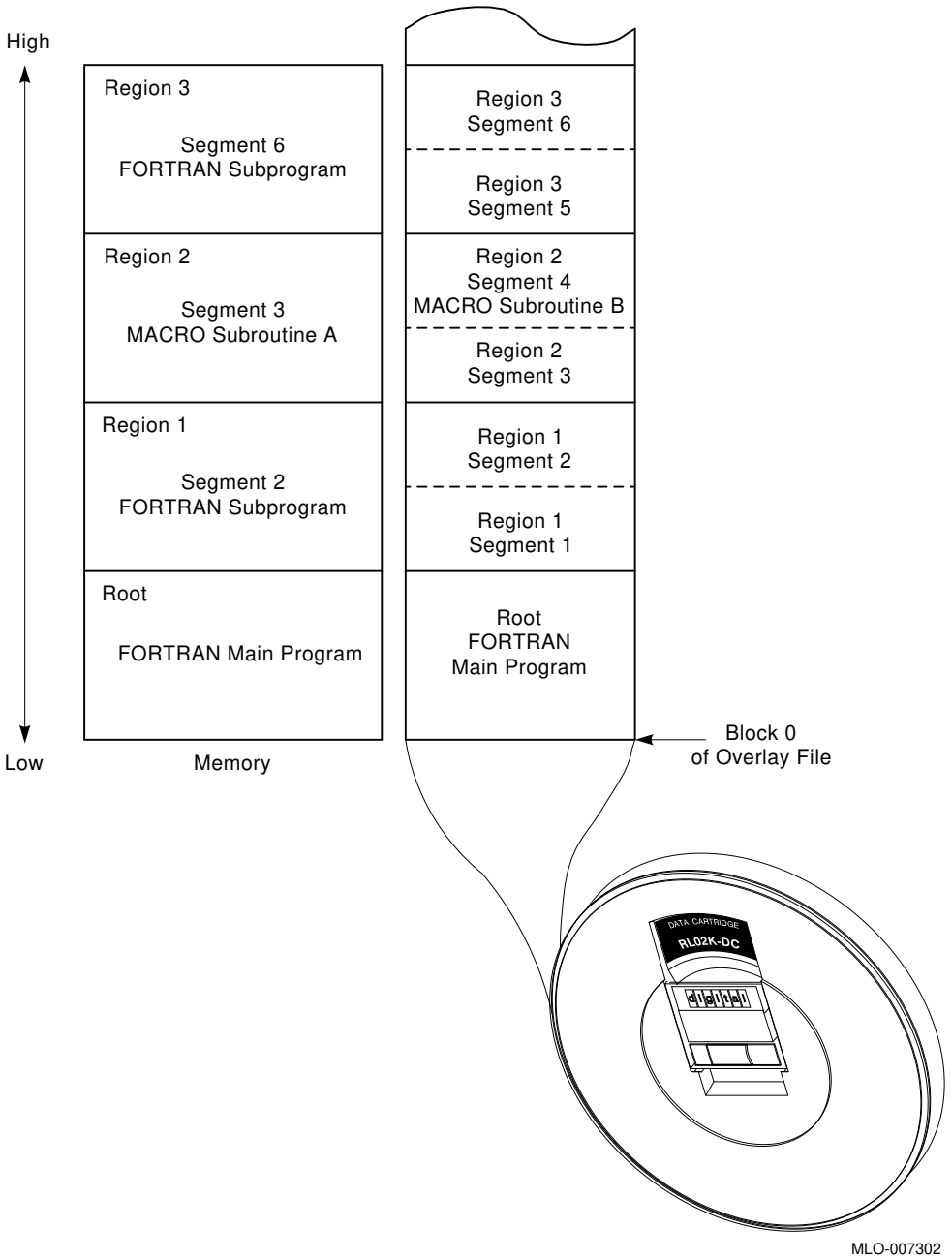
Low-Memory Overlays

An overlay structure divides a program into segments. For each overlaid program there is one root segment and a number of overlay segments. Each overlay segment is assigned to a particular area of available memory called an overlay region. More than one overlay segment can be assigned to a given overlay region. However, each region of memory is occupied by one (and only one) of its assigned segments at a time. The other segments assigned to that region are stored on disk or diskette. They are brought into memory when called, replacing (overlaying) the segment previously stored in that region. The root segment, on the other hand, contains those parts of the program that must always be memory resident. Therefore the root is never overlaid by another segment.

Figure A–1 diagrams an overlay structure for a FORTRAN program. The main program is placed in the root segment and is never overlaid. The various MACRO subroutines and FORTRAN subprograms are placed in overlay segments. Each overlay segment is assigned to an overlay region and stored on disk until called into memory. For example, region 2 is shared by the MACRO subroutine A currently in memory and the MACRO subroutine B in segment 4. When a call is made to subroutine B, segment 4 is brought into region 2 of memory, overlaying or replacing segment 3.

The overlay file, shown on the disk in Figure A–1, is created by the linker when you specify an overlay structure. The overlay file contains at all times a copy of the root segment and each overlay segment, including those overlay segments currently in memory.

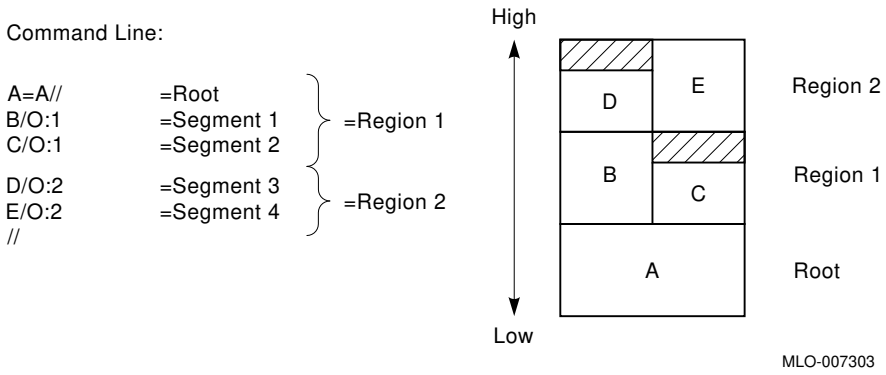
Figure A–1: Sample Overlay Structure for a FORTRAN Program



You specify an overlay structure to the linker by using the /O option (see Figure A–2). To specify an overlay structure that uses extended memory, use the /V option (see the Extended-Memory Overlays section for a discussion of extended-memory overlays). This option is described fully in the Extended-Memory Overlay Option (/V:n[:m]) section.

Low-Memory Overlays

Figure A–2: Overlay Scheme



The linker calculates the size of any region to be the size of the largest segment assigned to that region. Thus, to reduce the size of a program (that is, the amount of memory it needs), you should first concentrate on reducing the size of the largest segment in each region. The linker delineates the overlay regions you specify, and prefaces your program with the OHANDL overlay handler code shown in the next section. The linker also sets up links between the overlay handler and program references to routines that reside in overlays. When, at run time, a reference is made to a section of your program that is not currently in memory, these links cause an overlay to be read into memory. The overlay segment containing the referenced code becomes resident.

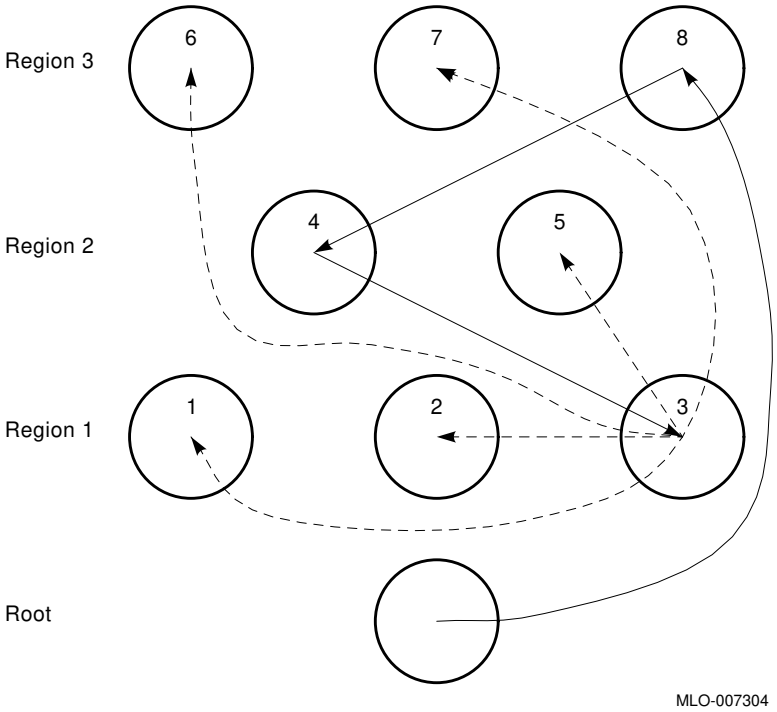
There is no special formula for creating an overlay structure. You do not need a special code or function call. However, some general guidelines must be followed. For example, a FORTRAN main program must always be placed in the root segment. This is true also for a global program section (such as a named COMMON block) that is referenced by more than one overlay segment.

The assignment of region numbers to overlay segments is crucial. Segments that overlay each other (have the same region number) must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region. Segments that need to be memory resident simultaneously must be assigned to different regions.

When you make calls to routines or subprograms that are in overlay segments, the entire return path must be in memory. This means that from an overlay segment you cannot call a routine that is in a different segment of the same region. If this is done, the called routine overlays the segment making the call and destroys the return path.

Figure A–3 illustrates a sample set of subroutine calls and return paths. In the example, solid lines represent valid subroutine calls and dotted lines represent invalid calls.

Figure A–3: Sample Subroutine Calls and Return Paths



Suppose the following subroutine calls were made:

1. The root calls segment 8
2. Segment 8 calls segment 4
3. Segment 4 calls segment 3

Segment 3 can now call any of the following, in any order:

Itself
Segment 4
Segment 8
The root

These segments and the root, of course, are all currently resident in memory.

Segment 3 cannot call any of the following segments because this would destroy its return path:

Segments 2 and 1
Segment 5
Segments 6 and 7

Look at what might happen if one of these invalid calls is made. Assume that segments 3, 4, and 5 all contain MACRO subroutines. Suppose segment 4 calls segment 3 and segment 3 in turn calls segment 5. Segment 5 is not resident in region 2, so an overlay read-in occurs: segment 5 is read into memory, thus destroying the

memory-resident copy of segment 4. The subroutine in segment 5 executes and returns control to segment 3. Segment 3 finishes its task and tries to return control to segment 4. Segment 4, however, has been replaced in memory by segment 5. Segment 4 cannot regain control and the program loops indefinitely, traps, or random results occur.

The guidelines already mentioned and some additional rules for creating overlay structures are summarized below:

- SYSLIB must be present to create an overlay structure because it contains the overlay handlers.
- Overlay segments assigned to the same region must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region.
- The root segment contains the transfer address, stack space, impure variables, data, and variables needed by many different segments. The FORTRAN main program unit must be placed in the root segment.
- A global program section (such as a named COMMON block or a .PSECT with the GBL attribute) that is referenced in more than one segment is placed in the root segment by the linker. This permits common access across the different segments.
- Object modules that are automatically acquired from a library file will automatically be placed in an overlay segment, so long as that library module is referenced only by that segment. If a library module is referenced by more than one segment, LINK places that library module in the root unless you use the /D option. See the Duplicate Global-Symbol Option (/D) section for more details on /D.

  Do not specify a library file on the same command line as an overlay segment. You must specify all library modules before specifying any overlay modules. Link places in the root any modules from a multiple-definition library and any modules included with the /I option.
- All COMMON blocks that are initialized with DATA statements must be similarly initialized in the segment in which they are placed.
- When you make calls to overlay segments, the entire return path to the calling routine must be in memory. (With extended-memory overlays, the entire return path must be mapped. See the Extended-Memory Overlays section.) This means you should take the following points into account:
  - — You can make calls with expected return (as from a FORTRAN main program to a FORTRAN or MACRO subroutine) from an overlay segment to entries in the same segment, the root segment, or to any other segment, so long as the called segment does not overlay in memory part of your return path to the main program.
  - — You can make jumps with no expected return (as in a MACRO program) from an overlay segment to any entry in the program with one exception: you

cannot make such a jump to a segment if the called segment will overlay an active routine (that is, a routine whose execution has begun, but not finished, and that will be returned to) in that region.

— Calls you make to entries in the same region as the calling routine must be entirely within the same segment, not within another segment in the same region.

- You must make calls or jumps to overlay segments directly to global symbols defined in an instruction PSECT (entry points). For example, if ENTER is a global tag in an overlay segment, the first of the following two commands is valid, but the second is not:

```
JMP ENTER       ;VALID
JMP ENTER+6     ;INVALID
```

- You can use globals defined in an instruction PSECT (entry points) of an overlay segment only for transfer of control and not for referencing data within an overlay segment. The assembler and linker cannot detect a violation of this rule so they issue no error. However, such a violation can cause the program to use incorrect data. If you reference these global symbols outside of their defining segment, the linker resolves them by using dummy subroutines of four words each in the overlay handler. Such a reference is indicated on the load map by a @ following the symbol.
- The linker directly resolves symbols that you define in a data PSECT. It is your responsibility to load the data into memory before referencing a global symbol defined in a data section.
- You cannot use a section name to pass control to an overlay because it does not load the appropriate segment into memory. For example, JSR PC,OVSEC is invalid if you use OVSEC as a .CSECT name in an overlay. You must use a global symbol to pass control from one segment to the next.
- In the linker command string, specify overlay regions in ascending order.
- Overlay regions are read-only. Unlike USR swapping, an overlay handler does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are reinitialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. You should place any variables or tables whose values must be maintained across overlays in the root segment.
- Your program cannot use channel 17_8 because overlays are read on that channel.
- MACRO and FORTRAN directly resolve all global symbols that are defined in a module. If LINK moves the PSECT where they are defined from an overlay segment to the root, LINK will not generate an overlay table entry for those symbols.

See the appropriate FORTRAN IV or FORTRAN–77 user's guide for additional information.

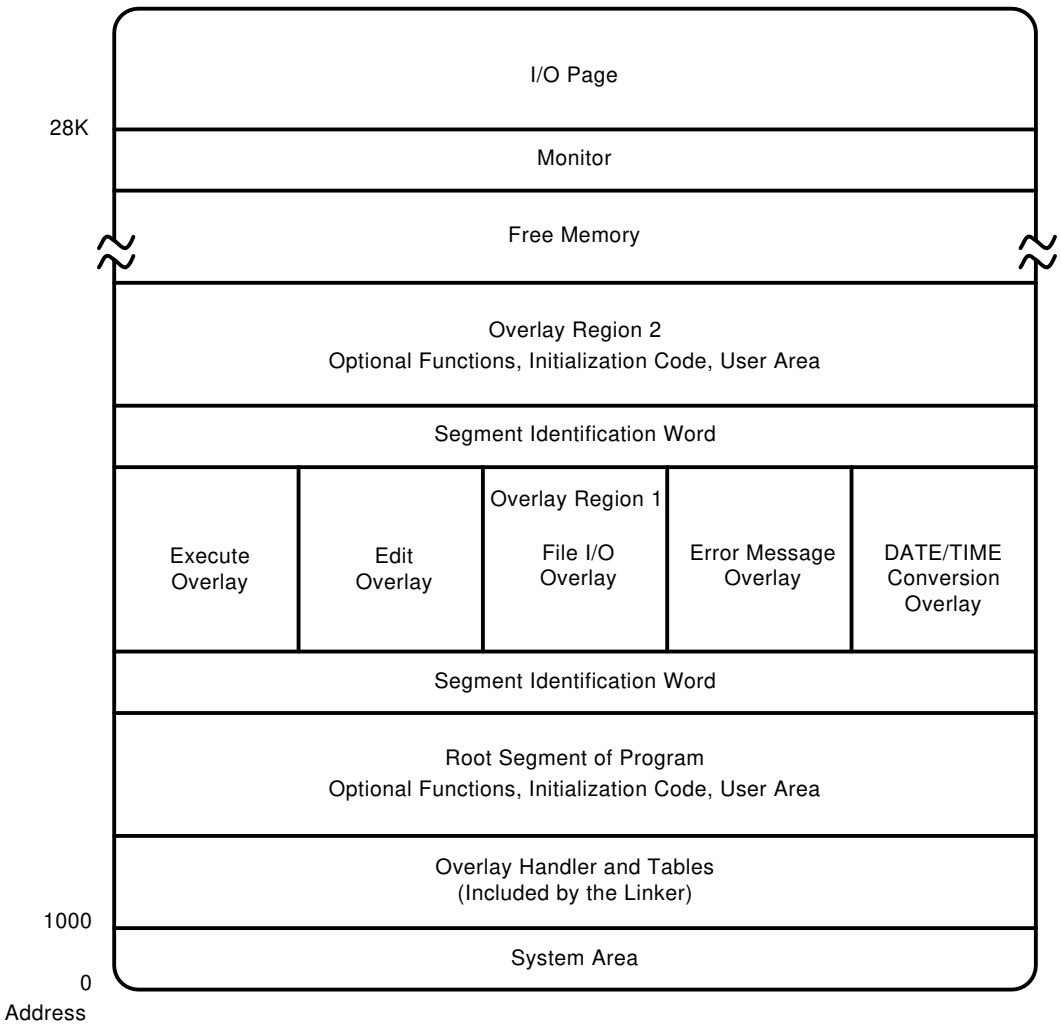
Low-Memory Overlays

The absolute section (. ABS.) never takes part in overlaying in any way. It is part of the root and is always resident.

This set of rules applies only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP–11 (as described in the *PDP–11 Processor Handbook* and in the FORTRAN and MACRO–11 language reference manuals). Note that the condition codes set by your program are not preserved across overlay segment boundaries.

The linker provides overlay services by including a small resident overlay handler in the same file with your program to be used at program run time. The linker inserts this overlay handler plus some tables into your program beginning at the bottom address. The linker then moves your program up in memory to make room for the overlay handler and tables, if necessary. The handler is stored in SYSLIB. This scheme is diagrammed in Figure A–4.

Figure A–4: Memory Diagram Showing BASIC Link with Overlay Regions



Low-Memory Overlay Handler (OHANDL)

```
.TITLE  OHANDL - Disk Only Overlay Handler
OVR$DK=1
OVR$MP=0
OVR$ID=0
OVR$XH=0
OVR$LO=0

.MCALL  .MODULE
.MODULE UHANDL,VERSION=16,COMMENT=<Universal Overlay Handler>....

;                          COPYRIGHT 1990, 1991 BY
;             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;                        ALL RIGHTS RESERVED.
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; the information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.

.SBTTL  Conditional Summary
.NLIST  CND
;+
;COND
;
;       OVR$DK  0       No /O overlays
;              (1)      Support /O overlays
;       OVR$MP  0       No /V overlays
;              (1)      Support /V overlays
;       OVR$ID  0       Support I=D environment
;              (1)      Support I<>D & Supy environments
;       OVR$XH (0)      Support multiple overlays
;               1       Support just 1 overlay ((X|Y)HANDL)
;       OVR$LO  0       Overlay handler inits /O area
;               1       Loader inits /O area
;              (OVR$ID) Defaults to same as I&D support
;
;
;                       OVR$DK or OVR$MP must be on (or both)
;                       OVR$XH forces not OVR$DK
;                       OVR$XH forces OVR$MP
;                       OVR$ID forces OVR$MP
;-
; MAS,SHD,LCP,DBB,JFW,DBB,JFW

.ASECT
.=32                    ;second word of EMT vector in IMAGE
.BYTE   OVR$DK+<2*OVR$MP>+<4*OVR$ID>+<10*OVR$XH>+<20*OVR$LO>
.BYTE   .UHAND

.VHAND  ==        .UHAND

.SBTTL  The Run-Time Overlay Handler
.DSABL  GBL
```

```
;+
; The following code is included in the user's program by the
; linker whenever low memory overlays are requested by the user.
; The run-time low memory overlay handler is called by a dummy
; subroutine of the following form:
;
;   .PSECT  $OTABL,D,GBL,OVR
;   JSR     R5,$OVRH          ;Call to common code for low memory
;                             ;overlays
;   .WORD   <OVERLAY # *6>    ;# of desired segment
;   .WORD   <ENTRY ADDRESS>   ;Actual core address (virtual address)
;
; One dummy routine of the above form is stored in the resident portion
; of the user's program for each entry point to a low memory overlay
; segment. All references to the entry point are modified by the linker
; to be references to the appropriate dummy routine.  Each overlay segment
; is called into core as a unit and must be contiguous in core.  An
; overlay segment may have any number of entry points, to the limits
; of core memory.  Only one segment at a time may occupy an overlay region.
;
;
; The segment number is an index into a table of overlay loading blocks.
; The entries for the /O form of overlays contain the following:
;
;   .PSECT $OTABL,D,GBL,OVR
;   .WORD  Segment start address
;   .WORD  File block number
;   .WORD  Word count
;
; There is one word prefixed to every overlay region that identifies the
; segment currently resident in that overlay region. This word is an index
; into the overlay table and points at the overlay segment information.
;
; Undefined globals in the overlay handler must be named "$OVDF1" to
; "$OVDFn" such that a range check may be done by LINK to determine if
; the undefined global name is from the overlay handler. A check is
; done on the .RAD50 characters "$OV", and then a range check is done on
; the .RAD50 charaters "DF1" to "DFn". These global symbols do not appear
; on link maps, since their value is not known until after the map has been
; printed.  Currently $OVDF1 to $OVDF6 are in use.
;
; Global symbols O$READ and O$DONE are useful when debugging overlaid
; programs.
;
;   O$READ will appear in the LINK map and locates the .READW
;   statement in the overlay handler.
;
;   O$DONE will appear in the LINK map and locates the first
;   instruction after the .READW in the overlay handler.
;
;-

.MCALL  .READW    ..V1..
.MCALL  .CKXX     .ASSUME      .BR
        ..V1..                       ;V1 format

        .CKXX   <R0,R1,R1A,R2,R2A,R5,R5A>
C.WDB=1234                  ;check value for WDB address
C.OVR=2234                  ;check value for overlay address
C.ONUM=60                   ;check value for overlay number
```

Low-Memory Overlay Handler (OHANDL)

```
.LIBRARY "SRC:SYSTEM"
.MCALL  .EMTDF    .ERRDF    ..READ
.MCALL  .OTBDF    .OTJDF    .OVRDF     .SYCDF     .UEBDF     .VTBDF
        .EMTDF              ;define EMT request numbers
        .ERRDF              ;define error numbers
        .OTBDF              ;overlay table entry definitions (/O)
        .OTJDF              ;overlay jump entry
        .OVRDF              ;overlay handle - SIPP communications
        .SYCDF              ;SYSCOM area
        .UEBDF              ;user error bit masks
        .VTBDF              ;overlay table entry definitions (/V)

OVRCHN  =:        17        ;overlay channel number

.GLOBL  $OVDF1
.GLOBL  $OVDF2
.GLOBL  $OVTAB
.WEAK   $OVTAB
.WEAK   O$READ
.WEAK   O$DONE
.WEAK   $ODF1
.WEAK   $ODF2

.SBTTL  Overlay Handler Code

.PSECT  $OHAND,I,GBL,CON
.PSECT  $OTABL,D,GBL,OVR
.PSECT  $OHAND

.ENABL  LSB

                            CK.R5=OTJ.JS+4
;+
; There is one entry point $OVRH for /O (low memory) overlays.
;-

$OVRH::    ;/O overlay entry point
                            CK.R5A=OTJ.JS+4
        MOV     R0,-(SP)    ;Save registers
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        ASRB    #1          ;First call?
        BCC     20$         ;No
                            ;Yes, (and flag is cleared now)
        MOV     $ODF1,R1    ;Start address for clear operation
 10$:   CMP     R1,$ODF2    ;Are we done?
        BHIS    20$         ;HIS -> done, or no /O overlays
        CLR     (R1)+       ;Clear all low memory overlay regions
        BR      10$

20$:
                            CK.R5A OTJ.OV
        MOV     @R5,R1              ;Pick up overlay number
        ADD     #$OVTAB-OTB.ES,R1   ;Calculate table address
                                    ;Adjusting for index value
                            CK.R1=VTB.WD
                            CK.R1A=OTB.AD
                            CK.R1 VTB.WD,+2
                            CK.R1A OTB.AD,+2
        MOV     (R1)+,R2            ;Get first arg. of overlay seg. entry
                            CK.R2=C.WDB
                            CK.R2A=C.OVR
60$:
                            CK.R2A C.OVR   ;@R2 is first word in overlay
                            CK.R5A OTJ.OV,+2
        CMP     (R5)+,@R2           ;Is overlay already resident?
```

```
        BEQ     80$                 ;Yes, branch to it
;+
; The .READW arguments are as follows:
; channel number, core address, length to read, relative block on disk.
; These are used in reverse order from that specified in the call.
;-
                            CK.R1 VTB.BK,+2
                            CK.R1 VTB.SZ
                            CK.R1A OTB.BK,+2
                            CK.R1A OTB.SZ
O$READ::
        .READW  OVRCHN,R2,@R1,(R1)+    ;Read from overlay file
O$DONE::BCS     90$                    ;Branch on error
80$:                                   ;Restore users registers
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
                            CK.R5A OTJ.AD
        MOV     @R5,R5                 ;Get entry address
        RTS     R5                     ;Enter overlay routine and
                                       ;restore user's R5

90$:
;+
;ERROR
        EMT     ...ERR                 ;System error 10 (OVERLAY I/O)
        .BYTE   0,ER.OVE
;This gets converted by the monitor into SERR -5 "Error reading an overlay"
;-

.DSABL  LSB

$ODF1:: .WORD    $OVDF1     ;High addr root segment + 2 (nxt avail)
$ODF2:: .WORD    $OVDF2     ;High addr /O overlays + 2 (nxt avail)
OVTAB:                      ;The following are required by SIPP
                                .Assume $ODF1 EQ OVTAB+OVR.ER
                                .Assume $ODF2 EQ OVTAB+OVR.EO

.SBTTL  $OVTAB    OVERLAY TABLE

;+
; Overlay Table Structure:
;
; LOC 64 ->     $OVTAB:
;               .WORD  <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O overlays
; LOC 66 ->     .WORD  <WDB ADDR>,<RELATIVE BLK>,<WORD COUNT>    /V overlays
;               Dummy subroutines for all overlay segments
; $ODF4 ->      Window definition blocks for extended memory overlays (/V)
; $ODF5 ->      Word after the end of the window definition blocks (/V)
;
;NOTE: description incomplete
;-

.PSECT  $OTABL
$OVTAB::                            ;first argument block if I-D version
.END
```

Extended-Memory Overlays

You can use LINK to create an overlay structure for your program that uses extended memory. Although you need a hardware configuration that includes a memory management unit to run a program that has overlays in extended memory, you can link it on any RT–11 system. Read the Low-Memory Overlays section before reading this section—much of the information contained in that section applies to extended-memory overlays as well.

Usually, you can convert an overlaid program to use extended memory without modifying the code. The extended-memory overlay handler and the keyboard monitor include all the programmed requests necessary to access extended memory. The overlay tables also include additional data used by these requests, so you can access extended memory automatically without using extended-memory programmed requests in your program.

The extended-memory overlay structure is different from the low-memory overlay structure in that extended-memory overlays can reside concurrently in extended memory. This allows for speedier execution because, once read in, your program requires fewer I/O transfers with the auxiliary mass storage volume. If all program data is resident, and the program is loaded, the program may be able to run without an auxiliary mass storage volume. However, you must observe the same restrictions with extended-memory overlays that apply to low-memory overlays, especially regarding return paths. This section describes how to create a program with overlays in extended memory and ends with an example of such a program.

NOTE

Overlays that reside in extended memory can contain impure data, but impure data is not automatically initialized each time a new overlay segment maps over a segment that contains impure data.

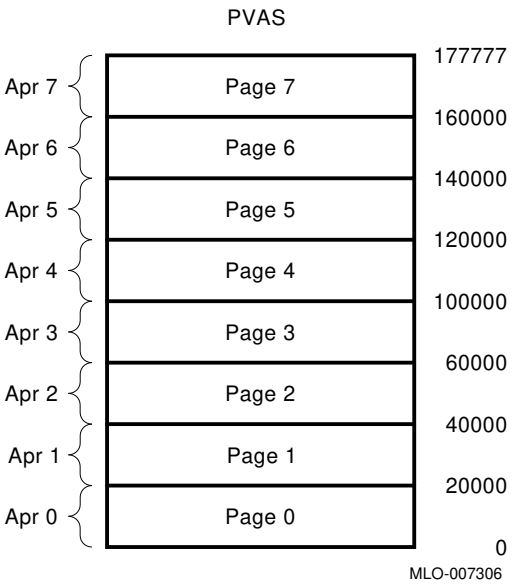
Virtual Address Space

When you set up an extended-memory overlay structure, you set it up as though you had locations 0 to 177777 (that is, 32K words of memory) available for your use. Physically, not all these locations are available to you in low memory; your program's absolute section resides, typically, in locations 0 to 500, and the monitor takes up a good deal of memory starting at location 160000, going downward. Also, the computer sets aside addresses 160000 to 177777 for the I/O page. But, because of memory management, you can structure your program as though you had all 32K words of memory for your use. This space is called the program virtual address space (PVAS). The memory management hardware and the monitor will allow part of your 32K address space to reside in extended memory.

The PVAS is divided into eight sections called pages, numbered 0-7. Each page contains 4K words. RT–11 references each page by the Active Page Register (APR). The APR contains the relocation constant, which controls the mapping for each page. Figure A–5 illustrates the PVAS, divided into pages. Keep in mind the structure of

your program in terms of how it uses the virtual address space so that you can design its overlay structure correctly and efficiently.

Figure A–5: Program Virtual Address Space



Each overlay that is to reside in extended memory must start on one of the 4K-word page boundaries. The linker automatically rounds up the size of each segment to achieve this. The linker thereby restricts you to a region reserved for the root, and a maximum of seven virtual overlay regions, each starting on a page boundary. If any of these segments extends beyond a 4K-word boundary, then one less virtual overlay region is available. For example, if the root is 5K words long, then the static region uses the addresses referenced by APRs 0 and 1. Only six virtual overlay regions will remain, those referenced by APRs 2 through 7.

Physical Address Space

When LINK creates the load module for a program that has overlays in extended memory, it defines how each overlay will be mapped to extended memory during run time. LINK handles extended-memory overlays differently from low-memory overlays. Figures A–6 and A–7 compare the differences.

Figure A–6 shows the physical address space of a program that has low memory overlays. Overlay segments share each region, and each overlay segment is read in from an auxiliary mass storage volume when called.

Figure A–6: Physical Address Space for Program with Low-Memory Overlays

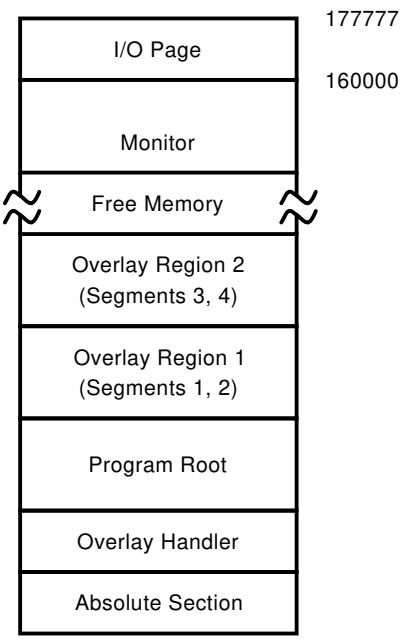
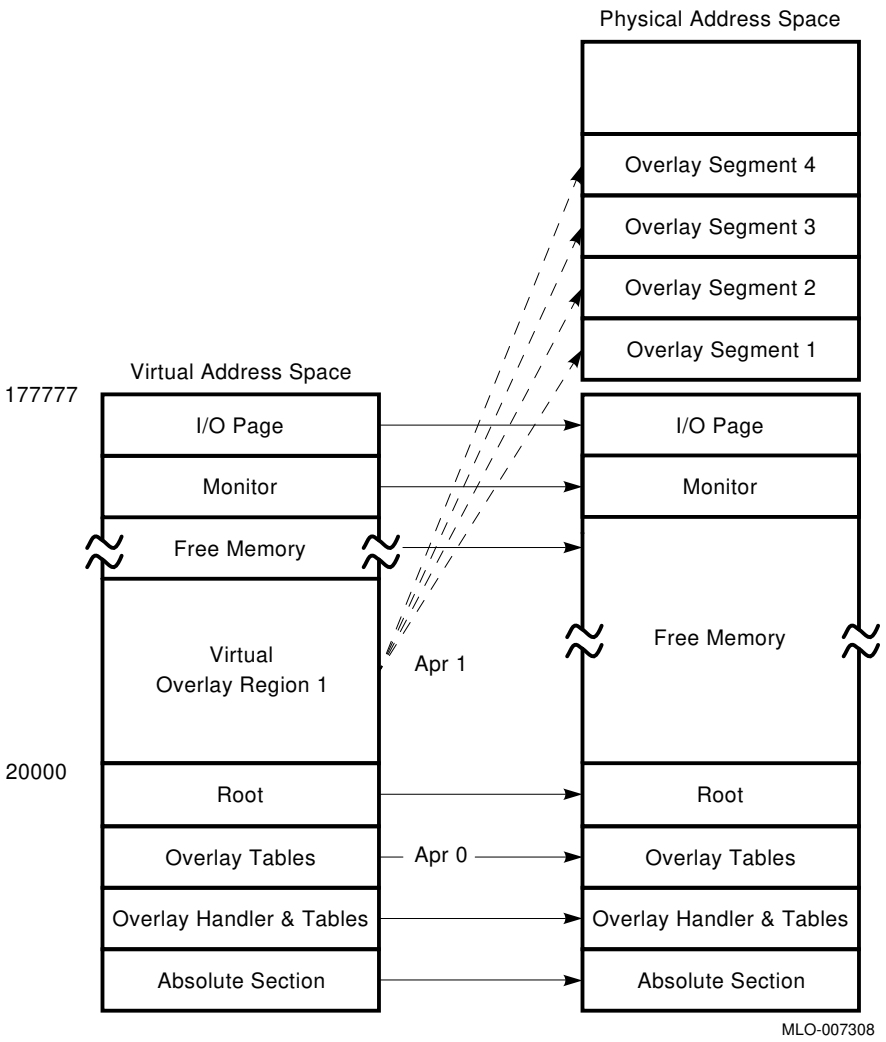


Figure A–7: Virtual and Physical Address Space



In Figure A–7, the diagram on the left shows the program virtual address space (0 to 177777). The diagram on the right shows the physical address space. In the program virtual address space, there is only one overlay region, and it starts on a 4K-word boundary (APR 1 references this region). The regions of address space that will map to extended memory are called virtual overlay regions. Notice the arrows that point from the virtual overlay region to a number of overlay segments that appear on the right.

The overlay segments in the virtual overlay region shown use the space specified by APR 1 (20000 to 37777), but they occupy contiguous areas of extended memory, called partitions. At run time, overlay segments 1 through 4, once called, are concurrently resident in extended memory, and no further disk I/O is done to access these segments.

Virtual and Privileged Jobs

The amount of virtual address space available to your program depends on the type of program you are running. Background, foreground, and system jobs can fall into two categories: virtual and privileged.

Virtual jobs can use all 32K words of the virtual address space, but they cannot directly access the I/O page, the monitor, the vectors, or other jobs. Unless you need to access these protected areas of memory, make your jobs virtual by setting bit 10 of the JSW.

Privileged jobs also have 32K words of virtual addressing space, but by default, the protected areas (monitor, I/O page, vectors, and so on) are part of this addressing space. Just as you may lose access to protected areas if you implement your own extended-memory mapping, you may lose access to the monitor and I/O page if you use extended-memory overlays with a privileged job.

Virtual and privileged jobs can map to extended memory. You can use extended-memory overlays with any type of virtual or privileged job (foreground, system, background).

Extended-Memory Overlay Load Map

The following is a sample load map for PROG.SAV, whose overlay structure is defined thus:

```
*PROG,PROG=MOD0//  RET
*MOD1/O:1  RET
*MOD2/O:1  RET
*MOD3/V:2  RET
*MOD4/V:3//  RET
```

The explanation following this map describes the portions of the load map devoted to low-memory and extended-memory overlays.

```
1     RT-11 LINK  V08.00      Load Map         Thursday 17-Jan-91 14:15  Page 1
2     V     .SAV      Title:  .MAIN.  Ident:
3
4     Section  Addr   Size    Global  Value    Global  Value   Global  Value
5
6      . ABS.  000000 001000 = 256.   words   (RW,I,GBL,ABS,OVR)
7      $OHAND  001000 000252 = 85.    words   (RW,I,GBL,REL,CON)
8                             $OVRHV  001000   $OVRH   001004  V$READ  001034
9                             V$DONE  001046   $VDF5   001234  $VDF4   001236
10                            $VDF1   001246   $VDF2   001250
11     $OTABL 001252 000114 = 38.    words   (RW,D,GBL,REL,OVR)
12            001366 000410 = 132.   words   (RW,I,LCL,REL,CON)
13     MAIN   001776 000070 = 28.    words   (RW,I,LCL,REL,CON)
14                            START   001776  RET1     002010  RET2    002014
15                            LIMIT   002024
16     LML4   002066 000026 = 11.    words   (RW,I,GBL,REL,CON)
17                            MSGL    002066
18     LML5   002114 000026 = 11.    words   (RW,I,GBL,REL,CON)
19                            MSGL2   002114
20     Segment size = 002142 = 561.  words
21
22    Overlay  region 000001 Segment 000001
23     LML2    002144 000032 = 13.   words   (RW,I,LCL,REL,CON)
24                            START1@ 002144
25     Segment size = 000032 = 13.   words
26
27    Overlay  region 000001  Segment 000002
28     LML3    002144 000036 = 15.   words   (RW,I,LCL,REL,CON)
29                             START2@ 002144
30     Segment size = 000036 = 15.   words
31
32
--------------------------------------------------------------------------
33
34     Virtual overlay region 000002
35     ----------------------
36
37     Partition      000001  Segment 000003
38      LML7   020002 000034 = 14.    words   (RW,I,LCL,REL,CON)
39                             START3  020002
40      LML6   020036 000042 = 17.    words   (RW,I,GBL,REL,CON)
41                             MSGL3   020036  RET4          020050
42     Segment size = 000076 = 31.    words
43
44     Virtual overlay region 000003
45     ----------------------
46
47     Partition  000002   Segment  000004
48     LML9      040002    000076 = 31.  words (RW,I,GBL,REL,CON)
49                                 MSGL9@ 040002
50     Segment size = 000076 = 31.   words
51
```

Extended-Memory Overlay Load Map

```
52
53      Transfer address = 001776, High limit = 002200 = 576.  words
54
55
56      Virtual high limit = 040076 = 8223.   words, next free address = 060000
57
58
59      Extended memory required = 000200 = 64.   words
60      RT-11   LINK   V08.00   Global Symbol Cross Reference Table   Page 1
61
62
63      $OVDF1 VHANDL+
64      $OVDF2 VHANDL+
65      $OVDF3 VHANDL+
66      $OVDF4 VHANDL+
67      $OVDF5 VHANDL+
68      $OVRH  VHANDL#+
69      $OVRHV VHANDL#+
70      $VDF1  VHANDL#+
71      $VDF2  VHANDL#+
72      $VDF4  VHANDL#+
73      $VDF5  VHANDL#+
74      LIMIT  .MAIN.#
75      MSGL   .MAIN.#
76      MSGL2  .MAIN.#
77      MSGL3  .MAIN.#
78      MSGL9  .MAIN.     .MAIN.#
79      RET1   .MAIN.#
80      RET2   .MAIN.#
81      RET4   .MAIN.#
82      START  .MAIN.#
83      START1 .MAIN.     .MAIN.#
84      START2 .MAIN.     .MAIN.#
85      START3 .MAIN.#
86      V$DONE VHANDL#+
87      V$READ VHANDL#+
```

Load Map Description

The following is a line-by-line description of the previous load map. This description refers to:

- Those portions of the load map that are unique to overlaid programs
- The global cross-reference table (which is not unique to overlaid programs).

For details on other parts of the load map, see the chapter on the LINK utility.

Line	Description
7-10	$OHAND PSECT. This is the overlay handler for overlays in both low and extended memory.
11	$OTABL PSECT. This program section contains tables of data used by the overlay handler.
12	Blank PSECT. The load map for overlaid programs lists the blank PSECT, when present, after the $OHAND and $OTABL PSECTs.
20	Contains data about the size of the program's root. The sections of the load map that follow provide information on the part of the program that is overlaid.

Line	Description
22	Header for overlay region 1, segment 1 (low memory overlay region).
23-24	LML2 PSECT. This is the only PSECT in segment 1. Notice in line 24 the @ character next to the global START1. This character indicates that its associated global is accessed through data contained in the overlay table PSECT, $OTABL, which is in the root.
25	Contains data on the size of segment 1.
32	Delineates the portion of the load map devoted to low memory from the portion devoted to extended memory.
34	Header for virtual overlay region 2. Note that overlay regions are numbered in ascending order, whether in low or extended memory.
37	Header for partition 1, segment 3.
41	Notice the absence of the @ character for the globals in PSECT LML6. This indicates that LML6 is not called outside segment 3.
42	Contains data on the size of overlay segment 3.
44	Header for virtual overlay region 3.
47	Header for partition 2, segment 4.
50	Contains data on the size of segment 4. Notice that segments 3 and 4 have the same length. LINK automatically rounds up the size of virtual overlay segments to multiples of 32_{10} words (or 100 octal bytes). LINK adds an overlay segment number word to the segment size number (the number 000076 that follows 040002 in line 48) to give the actual segment size.
53	Transfer address and high limit. The transfer address is the start address of the program. The high limit is the last low-memory address used by the root and unmapped overlays.
56	Virtual high limit. Indicates the last virtual address used by the part of the program in extended memory. The next free address is the address of the next page not in use by the program.
59	Indicates the amount of extended memory required by the program. Make sure you check this figure to ensure you have adequate space for your program at run time.
60-87	Cross-reference section of defined global symbols. Displays a cross-reference of all global symbols defined during the linking process. Note that global symbols are listed alphabetically and are followed by the names of the modules in which the global symbols are either defined or referenced. A pound sign (#) following a module name indicates that the global symbol is defined in that module. A plus sign (+) following a module name indicates that the module is from a library.

Extended-Memory Overlay Handler (VHANDL)

The following is the code for the extended-memory overlay handler.

```
.TITLE  VHANDL    - Disk and Mapped Overlay Handler
OVR$DK=1
OVR$MP=1
OVR$ID=0
OVR$XH=0
OVR$LO=0

.MCALL  .MODULE
.MODULE UHANDL,VERSION=16,COMMENT=<Universal Overlay Handler>,....

;                          COPYRIGHT 1990, 1991 BY
;             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;                        ALL RIGHTS RESERVED.
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; the information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.

.SBTTL  Conditional Summary
.NLIST  CND
;+
;COND
;
;       OVR$DK  0       No /O overlays
;              (1)      Support /O overlays
;       OVR$MP  0       No /V overlays
;              (1)      Support /V overlays
;       OVR$ID  0       Support I=D environment
;              (1)      Support I<>D & Supy environments
;       OVR$XH (0)      Support multiple overlays
;               1       Support just 1 overlay ((X|Y)HANDL)
;       OVR$LO  0       Overlay handler inits /O area
;               1       Loader inits /O area
;              (OVR$ID) Defaults to same as I&D support
;
;                       OVR$DK or OVR$MP must be on (or both)
;                       OVR$XH forces not OVR$DK
;                       OVR$XH forces OVR$MP
;                       OVR$ID forces OVR$MP
;-
; MAS,SHD,LCP,DBB,JFW,DBB,JFW

.ASECT
.=32                    ;second word of EMT vector in IMAGE
.BYTE   OVR$DK+<2*OVR$MP>+<4*OVR$ID>+<10*OVR$XH>+<20*OVR$LO>
.BYTE   .UHAND

.OHAND  ==        .UHAND
```

```
.SBTTL  The Run-Time Overlay Handler
.DSABL  GBL

;+
; The following code is included in the user's program by the
; linker whenever low memory overlays are requested by the user.
; The run-time low memory overlay handler is called by a dummy
; subroutine of the following form:
;
;       .PSECT $OTABL,D,GBL,OVR
;       JSR    R5,$OVRH        ;Call to common code for low memory overlays
;       .WORD  <OVERLAY # *6>  ;# of desired segment
;       .WORD  <ENTRY ADDRESS> ;Actual core address (virtual address)
;
; One dummy routine of the above form is stored in the resident portion
; of the user's program for each entry point to a low memory overlay
; segment. All references to the entry point are modified by the linker
; to be references to the appropriate dummy routine.  Each overlay segment
; is called into core as a unit and must be contiguous in core.  An
; overlay segment may have any number of entry points, to the limits
; of core memory.  Only one segment at a time may occupy an overlay region.
;
; If overlays in extended memory are specified, the following dummy
; subroutineis used as the entry point to the extended memory overlay
; handler.
;
;       .PSECT $OTABL,D,GBL,OVR
;       JSR    R5,$OVRHV               ;Call to common code for low memory
;                                      ;overlays
;       .WORD  <OVERLAY # *6>          ;# of desired segment
;       .WORD  <VIRTUAL ENTRY ADDRESS> ;Virtual address of segment
;
; The segment number is an index into a table of overlay loading blocks.
; The entries for the /O form of overlays contain the following:
;
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  Segment start address
;       .WORD  File block number
;       .WORD  Word count
;
; The entries for the /V form of overlays contain the following:
;
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  WDB address
;       .WORD  File block number
;       .WORD  Word count
;
; Additional data structures in the extended memory overlay handler and the
; overlay table permit use of extended memory. One region definition
; block is defined in the handler and XM EMTs are also included.  Window
; definition blocks for the extended memory partitions follow the dummy
; subroutines in the overlay table.
;
; There is one word prefixed to every overlay region that identifies the
; segment currently resident in that overlay region. This word is an index
; into the overlay table and points at the overlay segment information.
;
; Undefined globals in the overlay handler must be named "$OVDF1" to
; "$OVDFn" such that a range check may be done by LINK to determine if
; the undefined global name is from the overlay handler. A check is
; done on the .RAD50 characters "$OV", and then a range check is done on
; the .RAD50 charaters "DF1" to "DFn". These global symbols do not appear
; on link maps, since their value is not known until after the map has been
```

```
; printed.  Currently $OVDF1 to $OVDF6 are in use.
;
; Global symbols O$READ and O$DONE are useful when debugging overlaid
; programs.
;
;       O$READ will appear in the LINK map and locates the .READW
;       statement in the overlay handler.
;
;       O$DONE will appear in the LINK map and locates the first
;       instruction after the .READW in the overlay handler.
;
;-

.MCALL  .READW    ..V1..
.MCALL  .WDBDF    .RDBDF      .PRINT     .EXIT    .CRAW
.MCALL  .CKXX     .ASSUME     .BR
        ..V1..    ;V1 format
        .WDBDF    ;Define WDB offsets
        .RDBDF    ;Define RDB offsets

        .CKXX   <R0,R1,R1A,R2,R2A,R5,R5A>
C.WDB=1234                  ;check value for WDB address
C.OVR=2234                  ;check value for overlay address
C.ONUM=60                   ;check value for overlay number

.LIBRARY "SRC:SYSTEM"
.MCALL  .EMTDF    .ERRDF      ..READ
.MCALL  .OTBDF    .OTJDF      .OVRDF      .SYCDF   .UEBDF   .VTBDF
.MCALL  ..CRAW
        ..CRAW      ;.CRAW request offsets and values
        .EMTDF      ;define EMT request numbers
        .ERRDF      ;define error numbers
        .OTBDF      ;overlay table entry definitions (/O)
        .OTJDF      ;overlay jump entry
        .OVRDF      ;overlay handle -- SIPP communications
        .SYCDF      ;SYSCOM area
        .UEBDF      ;user error bit masks
        .VTBDF      ;overlay table entry definitions (/V)

OVRCHN  =:        17            ;overlay channel number

.GLOBL  $OVDF1
.GLOBL  $OVDF2
.GLOBL  $OVDF3
.GLOBL  $OVDF4
.GLOBL  $OVDF5
.WEAK   $OVTAB
.WEAK   O$READ
.WEAK   O$DONE
.WEAK   $ODF1
.WEAK   $ODF2
.WEAK   $ODF4
.WEAK   $ODF5

.SBTTL  Overlay Handler Code

.PSECT  $OHAND,I,GBL,CON
.PSECT  $OTABL,D,GBL,OVR
.PSECT  $OHAND

.ENABL  LSB
```

```
                                CK.R5=OTJ.JS+4
;+
; There are two entry points to the overlay handler: $OVRHV for /V
; (extended memory) overlays, and $OVRH for /O (low memory) overlays.
;-

$OVRHV::                        ;/V overlay entry point
        INCB    (PC)+           ;Set /V overlay entry switch
VFLAG:  .WORD     0             ;=0 IF /O ; =1 if /V overlay entry
$OVRH::                         ;/O overlay entry point
                              CK.R5A=OTJ.JS+4
        MOV     R0,-(SP)        ;Save registers
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        ASRB    #1              ;First call?
        BCC     20$             ;No
                                ;Yes, (and flag is cleared now)
        MOV     $ODF1,R1        ;Start address for clear operation
10$:    CMP     R1,$ODF2        ;Are we done?
        BHIS    20$             ;HIS -> done, or no /O overlays
        CLR     (R1)+           ;Clear all low memory overlay regions
        BR      10$

20$:
                               CK.R5 OTJ.OV
                               CK.R5A OTJ.OV
        MOV     @R5,R1             ;Pick up overlay number
        ADD     #$OVTAB-OTB.ES,R1 ;Calculate table address
                                   ;Adjusting for index value
                               CK.R1=VTB.WD
                               CK.R1A=OTB.AD
                               CK.R1 VTB.WD,+2
                               CK.R1A OTB.AD,+2
        MOV     (R1)+,R2       ;Get first arg. of overlay seg. entry
                               CK.R2=C.WDB
                               CK.R2A=C.OVR
        TSTB    VFLAG          ;Is this /V entry?
        BEQ     60$            ;If non-zero then no

;+
; Virtual overlay segments in the same region but in different
; partitions use different WDBs.  Only one of these windows exists
; at any time.  This is because when a new window in a virtual
; overlay region is created, the monitor implicitly eliminates
; any window that exists in that virtual overlay region.  Thus,
; if the called overlay segment is not currently mapped, its window
; must be re-created (.CRAWed) besides being mapped.  The mapping
; is done implicitly in the following code since the WS.MAP bit is
; set in all of the virtual overlay segments' WDBs.
;-
                                CK.R2 C.WDB
                                .Assume W.NID EQ 0
        TSTB    @R2     ;Do we need to create a window (.CRAW)?
        BEQ     30$     ;Yes
                                CK.R2 C.WDB
        MOV     @W.NBAS(R2),R0        ;Get index of segment now mapped
                                CK.R0=C.ONUM
        BEQ     30$                   ;There isn't one; we must .CRAW
                                CK.R0 C.ONUM
                                .Assume VTB.WD EQ 0
                                CK.R2 C.WDB
        CMP     $OVTAB-OTB.ES(R0),R2 ;Is overlay region same as this one?
        BEQ     50$                  ;If equal, just worry about disk I/O
30$:
```

```
        .CRAW   #AREA,R2,CODE=NOSET  ;Do the EMT for .CRAW
        BCS     100$                 ;Carry set means error!
50$:
                                CK.R2 C.WDB
        MOV     W.NBAS(R2),R2        ;Get memory address of overlay
                                CK.R2=C.OVR
60$:
                                CK.R2A C.OVR ;@R2 is first word in overlay
                                CK.R5 OTJ.OV,+2
                                CK.R5A OTJ.OV,+2
        CMP     (R5)+,@R2            ;Is overlay already resident?
        BEQ     80$                  ;Yes, branch to it

;+
; The .READW arguments are as follows:
; channel number, core address, length to read, relative block on disk.
; These are used in reverse order from that specified in the call.
;-
                                CK.R1 VTB.BK,+2
                                CK.R1 VTB.SZ
                                CK.R1A OTB.BK,+2
                                CK.R1A OTB.SZ
O$READ::
        .READW  OVRCHN,R2,@R1,(R1)+ ;Read from overlay file
O$DONE::BCS     90$                 ;Branch on error
80$:                                ;Restore users registers
        CLRB    VFLAG               ;Clear /V flag
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
                                CK.R5 OTJ.AD
                                CK.R5A OTJ.AD
        MOV     @R5,R5      ;Get entry address
        RTS     R5          ;Enter overlay routine and restore user's R5

90$:
;+
;ERROR
        EMT     ...ERR      ;System error 10 (OVERLAY I/O)
        .BYTE   0,ER.OVE
;This get converted by the monitor into SERR -5 "Error reading an overlay"
;-

; ERROR MESSAGE

100$:   MOV        #MSG2,R0        ;Otherwise error
        .PRINT                     ;And print message
        BISB    #<FATAL$>,@#$USRRB ;Set error byte
        .EXIT                      ;And exit

.DSABL  LSB

.ENABL  LC
.NLIST BEX
;+
;ERROR
MSG2:   .ASCIZ     /?VHANDL-F-Window error/
;-
        .EVEN

.LIST BEX

AREA:   .WORD      .CRAW*^O400+..CRAW,$OVDF4 ;EMT area block for .CRAW
                               .Assume .-AREA GE L.CRAW
```

```
$ODF5:: .WORD   $OVDF5     ;Pointer to word after WDBs in overlay table
$ODF4:: .WORD   $OVDF4     ;Pointer to start of WDBs in overlay table

RGADR:  .WORD     0        ;Three word region definition block
RGSIZ:  .WORD     $OVDF3,0 ;$OVDF3 set by LINK = size of region
                           .Assume RGADR+R.GSIZ EQ RGSIZ


$ODF1:: .WORD    $OVDF1    ;High addr root segment + 2 (nxt avail)
$ODF2:: .WORD    $OVDF2    ;High addr /O overlays + 2 (nxt avail)
OVTAB:                     ;The following are required by SIPP
                                .Assume $ODF5 EQ OVTAB+OVR.EW
                                .Assume $ODF4 EQ OVTAB+OVR.SW
                                .Assume RGSIZ EQ OVTAB+OVR.RS
                                .Assume $ODF1 EQ OVTAB+OVR.ER
                                .Assume $ODF2 EQ OVTAB+OVR.EO

.SBTTL  $OVTAB    OVERLAY TABLE

;+
; Overlay Table Structure:
;
; LOC 64 ->    $OVTAB:
;              .WORD  <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O overlays
; LOC 66 ->    .WORD  <WDB ADDR>,<RELATIVE BLK>,<WORD COUNT>    /V overlays
;              Dummy subroutines for all overlay segments
; $ODF4 ->     Window definition blocks for extended memory overlays (/V)
; $ODF5 ->     Word after the end of the window definition blocks (/V)
;
;NOTE: description incomplete
;-

.PSECT  $OTABL
$OVTAB::                              ;first argument block if I-D version
.END
```

Low- and Extended-Memory Overlays

You can combine low-memory overlays and extended-memory overlays in the same program structure. If you do so, however, each low-memory overlay region you use makes your remaining virtual address space smaller.

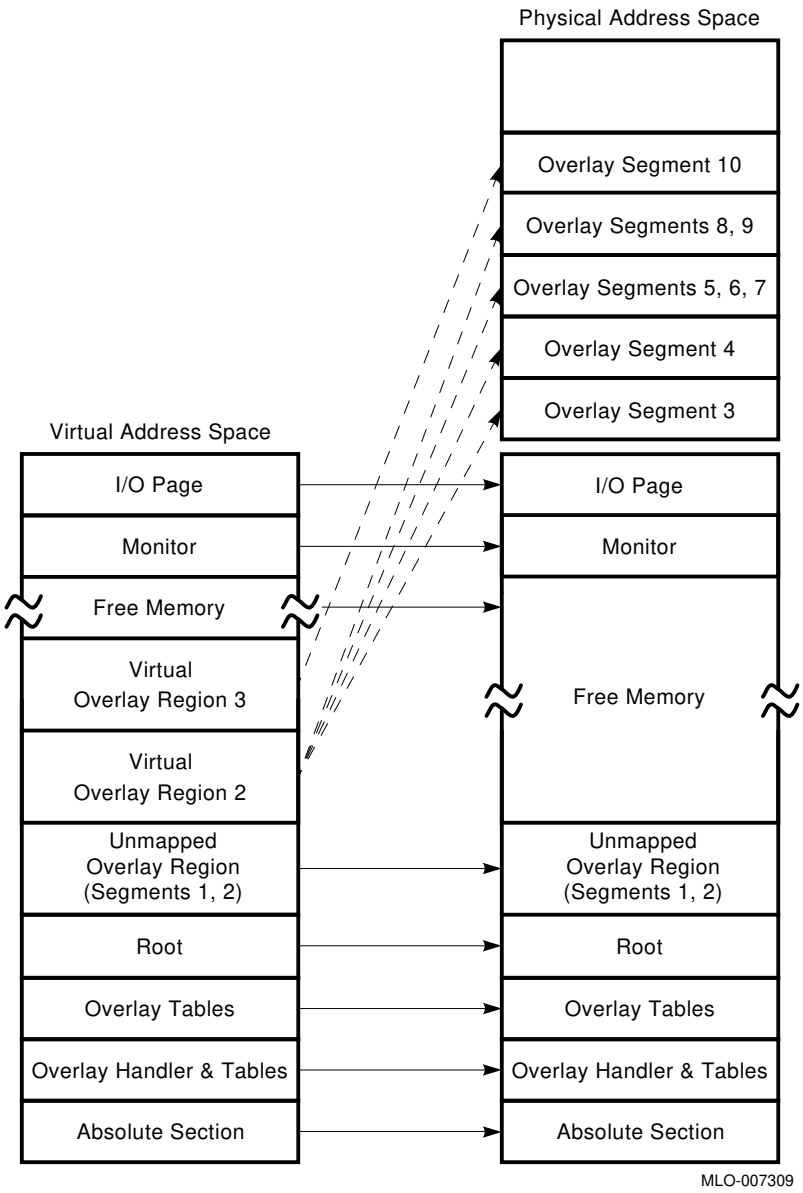
It is important to note that as you combine low-memory overlays with extended-memory overlays, you must list your regions in ascending order, whether or not one is a low-memory overlay region and the next is a virtual region. That is, if the first overlay region is a low-memory overlay region, specify it as region 1. If the next region is a virtual region, specify it as region 2. Note that you must specify low-memory overlays before extended-memory overlays.

The following example creates a low-memory overlay region and a virtual overlay region above it.

```
.R LINK  RET
*PROG=PROG//  RET
*SEG1/O:1  RET
*SEG2/O:1  RET
*SEG3/V:2  RET
*SEG4/V:2  RET
*SEG5/V:2:1  RET
*SEG6/V:2:1  RET
*SEG7/V:2:1  RET
*SEG8/V:2:2  RET
*SEG9/V:2:2  RET
*SEG10/V:3//  RET
```

Figure A–8 shows how low memory and extended memory might appear if the program from this example were loaded.

Figure A–8: Memory Diagram Showing Low-Memory and Extended-Memory Overlays



One Virtual Overlay Segment

XHANDL is a pseudo overlay handler included in the distributed system library, SYSLIB.OBJ. XHANDL is useful for programs that can be organized with a small root and one virtual overlay segment.

XHANDL is an *overlay handler* in that it contains just enough code to map a program's single virtual overlay segment to high memory. XHANDL is a *pseudo* overlay handler in that it lacks the capability of manipulating multiple overlays.

XHANDL uses less low memory than VHANDL (the virtual overlay handler), making more of that memory available to other handlers and jobs. You can save approximately 64_{10} words in low memory for every job that uses XHANDL instead of VHANDL.

XHANDL does the following:

1. Creates a region in extended memory.
2. Loads an overlay segment into that region.
3. Maps the region.
4. Calls your program's root.

Then, your program can jump (using the JMP instruction) to the overlay region.

The following RT–11 V5.5 system utilities use XHANDL:

- INDEX
- KEX
- VTCOM
- SPOOL

If you want to create a program that uses XHANDL, you can do it in either of the following two ways. Method A uses XHANDL in SYSLIB. Method B has you *extract* XHANDL from SYSLIB and *link* it with your root. Method B lets you use a smaller root than Method A:

METHOD A: Using XHANDL in SYSLIB

1. Create or prepare your root program; for example:

```
.SBTTL TESTX.MAC

;+
; This is the root of an example program for testing
; XHANDL.
;-

.MCALL  .EXIT, .PRINT

        JSW     =       44      ;Job Status Word
        VIRT$   =       2000    ;Virtual bit in JSW
```

```
        .ASECT
        .=JSW
        .WORD   VIRT$           ;Set virtual bit in JSW
        .PSECT
        .GLOBL  OVLYX           ;Start address in OVLYX
START:: NOP
        .PRINT  #HERE
        JMP     OVLYX           ;Call the overlay

BACK::  .PRINT  #BYE            ;Show return to root
        .EXIT

HERE:   .ASCIZ /We are in the ROOT/
BYE:    .ASCIZ /Exiting TESTX/
        .EVEN
        .END    START
```

2. Create your overlay segment; for example, the following overlay segment, OVLYX.MAC, can be used with the preceding root program, TESTX.MAC:

```
.SBTTL OVLYX.MAC

;+
; This is a single overlay example for using XHANDL.
;-

        .MCALL  .PRINT, .EXIT
        .GLOBL  BACK

OVLYX:: .PRINT  #HERE
        JMP     BACK

HERE:   .ASCIZ /We are in the OVERLAY/
        .EVEN
        .END
```

3. Compile your program and your overlay segment; for example:

```
.MAC TESTX RET

.MAC OVLYX RET

.
```

4. Link your program with your overlay segment.

   Use the following LINK command with the /INCLUDE, /PROMPT, and /XM input options and respond to the *Library search?* prompts as indicated:

```
.LINK root/INCLUDE/PROMPT/XM RET
*overlay/V:1// RET
Library search? $OVRHX RET
Library search? RET

.
```

For example:

```
.R LINK  RET
*TESTX/V,TESTX=TESTX/I//  RET
*OVLYX/V:1//  RET
Library search? $OVRHX  RET
Library search?  RET
* CTRL/C

.
```

or:

```
.LINK/MAP:TESTX TESTX/INCLUDE/PROMPT/XM  RET
*OVLYX/V:1//  RET
Library search? $OVRHX  RET
Library search?  RET

.
```

5. Run your program; for example:

```
.RUN TESTX  RET
We are in the ROOT
We are in the OVERLAY
Exiting TESTX

.
```

METHOD B: Extracting XHANDL from SYSLIB

1. Create or prepare your root program; for example:

```
.SBTTL TESTX1.MAC

;+
; This is the root of an example program for testing
; XHANDL.
;
;-
        JSW     =       44      ;Job Status Word
        VIRT$   =       2000    ;Virtual bit in JSW

        .ASECT
        .=JSW
        .WORD   VIRT$           ;Set virtual bit in JSW
        .PSECT
        .GLOBL  OVLYX1          ;Start address in OVLYX1
        .END
```

2. Create or prepare your overlay segment; for example:

```
.SBTTL OVLYX1.MAC

;+
; This is the single overlay for using XHANDL
;-
        .MCALL  .PRINT, .EXIT

OVLYX1::.PRINT  #HERE
        .EXIT
```

```
HERE:   .ASCIZ /We are in the OVERLAY/
        .EVEN
        .END
```

3. Extract XHANDL from SYSLIB. The LIBR utility prompts you for the second, third, and fourth line. To end the command, press RETURN; for example:

```
.LIBRARY/EXTRACT [RET]
Library? SYSLIB.OBJ [RET]
File? XHANDL.OBJ [RET]
Global? $OVRHX [RET]
Global? [RET]
```

.

4. Compile your program and your overlay segment; for example:

```
.MAC TESTX1 [RET]
```

```
.MAC OVLYX1 [RET]
```

.

5. Link your program with the segment; for example:

```
.LINK/MAP:TESTX1 TESTX1,XHANDL/PROMPT   [RET]
*OVLYX1/V:1 [RET]
*// [RET]
```

.

or:

```
.R LINK [RET]
*TESTX1,TESTX1=TESTX1,XHANDL//   [RET]
*OVLYX1/V:1// [RET]
* [CTRL/C]
```

.

6. Run your program; for example:

```
.RUN TESTX1 [RET]
We are in the OVERLAY
```

.

Pseudo Overlay Handler (XHANDL)

```
.TITLE  XHANDL    - Single Mapped Overlay Handler
OVR$DK=0
OVR$MP=1
OVR$ID=0
OVR$XH=1
OVR$LO=0

.MCALL  .MODULE
.MODULE UHANDL,VERSION=16,COMMENT=<Universal Overlay Handler>,....

;                          COPYRIGHT 1990, 1991 BY
;             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;                        ALL RIGHTS RESERVED.
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; the information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.

.SBTTL  Conditional Summary
.NLIST  CND
;+
;COND
;
;       OVR$DK  0       No /O overlays
;              (1)      Support /O overlays
;       OVR$MP  0       No /V overlays
;              (1)      Support /V overlays
;       OVR$ID  0       Support I=D environment
;              (1)      Support I<>D & Supy environments
;       OVR$XH (0)      Support multiple overlays
;               1       Support just 1 overlay ((X|Y)HANDL)
;       OVR$LO  0       Overlay handler inits /O area
;               1       Loader inits /O area
;              (OVR$ID) Defaults to same as I&D support
;
;                      OVR$DK or OVR$MP must be on (or both)
;                      OVR$XH forces not OVR$DK
;                      OVR$XH forces OVR$MP
;                      OVR$ID forces OVR$MP
;-
; MAS,SHD,LCP,DBB,JFW,DBB,JFW

OVR$DK=0
OVR$MP=1

.ASECT
.=32                    ;second word of EMT vector in IMAGE
.BYTE   OVR$DK+<2*OVR$MP>+<4*OVR$ID>+<10*OVR$XH>+<20*OVR$LO>
.BYTE   .UHAND

.XHAND  ==        .UHAND
```

```
.SBTTL  The Run-Time Overlay Handler
.DSABL  GBL

;++
; This version of the overlay handler can be used only for jobs that have
; a dummy root and exactly one virtual overlay.
;--

; If overlays in extended memory are specified, the following dummy
; subroutine is used as the entry point to the extended memory overlay
; handler.
;
;       .PSECT $OTABL,D,GBL,OVR
;       JSR    R5,$OVRHV      ;Call to common code for low memory overlays
;       .WORD  <OVERLAY # *6>            ;# of desired segment
;       .WORD  <VIRTUAL ENTRY ADDRESS>   ;Virtual address of segment
;
; The entries for the /V form of overlays contain the following:
;
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  WDB address
;       .WORD  File block number
;       .WORD  Word count
;
; Additional data structures in the extended memory overlay handler and
; the overlay table permit use of extended memory. One region definition
; block is defined in the handler and XM EMTs are also included.  Window
; definition blocks for the extended memory partitions follow the dummy
; subroutines in the overlay table.
;
; There is one word prefixed to every overlay region that identifies the
; segment currently resident in that overlay region. This word is an index
; into the overlay table and points at the overlay segment information.
;
; Undefined globals in the overlay handler must be named "$OVDF1" to
; "$OVDFn" such that a range check may be done by LINK to determine if
; the undefined global name is from the overlay handler. A check is
; done on the .RAD50 characters "$OV", and then a range check is done on
; the .RAD50 charaters "DF1" to "DFn". These global symbols do not appear
; on link maps, since their value is not known until after the map has
; been printed.  Currently $OVDF1 to $OVDF6 are in use.
;
; Global symbols O$READ and O$DONE are useful when debugging overlaid
; programs.
;
;       O$READ will appear in the LINK map and locates the .READW
;       statement in the overlay handler.
;
;       O$DONE will appear in the LINK map and locates the first
;       instruction after the .READW in the overlay handler.
;
; To use (X|Y)HANDL, create a root module of the form
;
;NOTE: YHANDL is not (yet) implemented
;
;       JSW    =   44          ;Job Status Word
;       VIRT$  = 2000          ;Virtual bit in JSW
;       .ASECT
;       .=JSW                  ;Set location counter to JSW
;       .WORD  VIRT$           ;Set virtual bit in JSW
;       .PSECT
;       .GLOBL start           ;Start address in overlay
;       .END
```

Pseudo Overlay Handler (XHANDL)

```
;
;
; Then use the following LINK command
;       LINK root/INCLUDE/PROMPT
;       overlay(s)/V:1:1//
;       Library search? $OVRH(X|Y)
;       Library search? <CR>
;-

.MCALL  .READW    ..V1..
.MCALL  .WDBDF    .RDBDF      .PRINT     .EXIT    .CRAW
.MCALL  .CKXX     .ASSUME     .BR
        ..V1..                ;V1 format
        .WDBDF                ;Define WDB offsets
        .RDBDF                ;Define RDB offsets

        .CKXX   <R0,R1,R1A,R2,R2A,R5,R5A>
C.WDB=1234                    ;check value for WDB address
C.OVR=2234                    ;check value for overlay address
C.ONUM=60                     ;check value for overlay number

.LIBRARY "SRC:SYSTEM"
.MCALL  .EMTDF    .ERRDF      ..READ
.MCALL  .OTBDF    .OTJDF      .OVRDF     .SYCDF    .UEBDF    .VTBDF
.MCALL  ..CRAW
        ..CRAW      ;.CRAW request offsets and values
        .EMTDF      ;define EMT request numbers
        .ERRDF      ;define error numbers
        .OTBDF      ;overlay table entry definitions (/O)
        .OTJDF      ;overlay jump entry
        .OVRDF      ;overlay handle -- SIPP communications
        .SYCDF      ;SYSCOM area
        .UEBDF      ;user error bit masks
        .VTBDF      ;overlay table entry definitions (/V)

OVRCHN  =:        17            ;overlay channel number

.GLOBL  $OVDF1
.GLOBL  $OVDF2
.GLOBL  $OVDF3
.GLOBL  $OVDF4
.GLOBL  $OVDF5
.WEAK   $OVTAB
.WEAK   O$READ
.WEAK   O$DONE
.WEAK   $ODF1
.WEAK   $ODF2
.WEAK   $ODF4
.WEAK   $ODF5

.SBTTL  OVERLAY HANDLER CODE

.PSECT  $OHAND,I,GBL,CON
.PSECT  $OTABL,D,GBL,OVR
.PSECT  $OHAND

.ENABL  LSB
```

```
$OVRHX::        ;Global used to load this variant from library
        .WEAK   $OVRHV
$OVRHV::                            ;/V overlay entry point
        .CRAW   #AREA,CODE=NOSET    ;Do the EMT for .CRAW
        BCS     90$                 ;Carry set means error!
        MOV     #$OVTAB,R1          ;Load table address
                                  CK.R1=VTB.WD
                                  CK.R1 VTB.WD,+2
        MOV     (R1)+,R2    ;Get first arg. of overlay seg. entry
                                  CK.R2=C.WDB
                                  CK.R2 C.WDB
        MOV     W.NBAS(R2),R2       ;Get memory address of overlay
                                  CK.R2=C.OVR
60$:

;+
; The .READW arguments are as follows:
; channel number, core address, length to read, relative block on disk.
; These are used in reverse order from that specified in the call.
;-
                                  CK.R1 VTB.BK,+2
                                  CK.R1 VTB.SZ
O$READ::
        .READW  OVRCHN,R2,@R1,(R1)+ ;Read from overlay file
O$DONE::BCS     90$    ;Branch on error
;>>>number?
80$:
        JMP     @$OVTAB+14   ;Go to first entry address

90$:
100$:
;+
;ERROR
        EMT     ...ERR       ;System error 10 (OVERLAY I/O)
        .BYTE   0,ER.OVE
;This get converted by the monitor into SERR -5 "Error reading an overlay"
;-

.DSABL  LSB

.SBTTL IMPURE AREA

AREA:   .WORD       .CRAW*^O400+..CRAW,$OVDF4 ;EMT area block for .CRAW
                                  .Assume .-AREA GE L.CRAW

$ODF5:: .WORD   $OVDF5    ;Pointer to word after WDBs in overlay table
$ODF4:: .WORD   $OVDF4    ;Pointer to start of WDBs in overlay table

RGADR:  .WORD   0         ;Three word region definition block
RGSIZ:  .WORD   $OVDF3,0  ;$OVDF3 set by LINK = size of region
                          .Assume RGADR+R.GSIZ EQ RGSIZ


$ODF1:: .WORD   $OVDF1    ;High addr root segment + 2 (nxt avail)
$ODF2:: .WORD   $OVDF2    ;High addr /O overlays + 2 (nxt avail)
OVTAB:             ;The following are required by SIPP
                          .Assume $ODF5 EQ OVTAB+OVR.EW
                          .Assume $ODF4 EQ OVTAB+OVR.SW
                          .Assume RGSIZ EQ OVTAB+OVR.RS
                          .Assume $ODF1 EQ OVTAB+OVR.ER
                          .Assume $ODF2 EQ OVTAB+OVR.EO

.SBTTL  $OVTAB  OVERLAY TABLE
```

Pseudo Overlay Handler (XHANDL)

```
;+
; OVERLAY TABLE STRUCTURE:
;
; LOC 64 ->  $OVTAB:
;            .WORD  <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O overlays
; LOC 66 ->  .WORD  <WDB ADDR>,<RELATIVE BLK>,<WORD COUNT>    /V overlays
;            Dummy subroutines for all overlay segments
; $ODF4 ->   Window definition blocks for extended memory overlays (/V)
; $ODF5 ->   Word after the end of the window definition blocks (/V)
;
;NOTE: description incomplete
;-

.PSECT  $OTABL
$OVTAB::                  ;first argument block if I-D version
.END    $OVRHX
```

Separate I and D Space Overlays

The LINK utility allows linking SAV image programs with separate I (Instruction) and D (Data) space.

User Jobs

User jobs can be written and built specially for use of separated I and D space. When writing code that makes use of separated I and D space, follow these guidelines:

- Code and data should be split up into I and D PSECTs, respectively.
- Code cannot be moved onto the stack for execution.
- Interrupt service routines are not supported in separated I and D space programs, since such programs are completely outside of kernel memory.
- Code and data go into one overlay segment pair that is composed of an I-space segment and a D-space segment. The D-space segment can be empty, but the I-space segment cannot be empty. The overlay handler loads both the I- and D-space segments of an overlay segment (unless the D-space segment is empty).
- Code and/or data can be forced into the root via the SAV PSECT attribute. This is especially desirable when you need to overlay code or data, but not both.
- There are new programmed requests specially for separated I and D space jobs.
- There are new arguments added to existing programmed requests specially for separated I and D space jobs.
- Separated I-D space cannot be used in .REL jobs or privileged jobs.

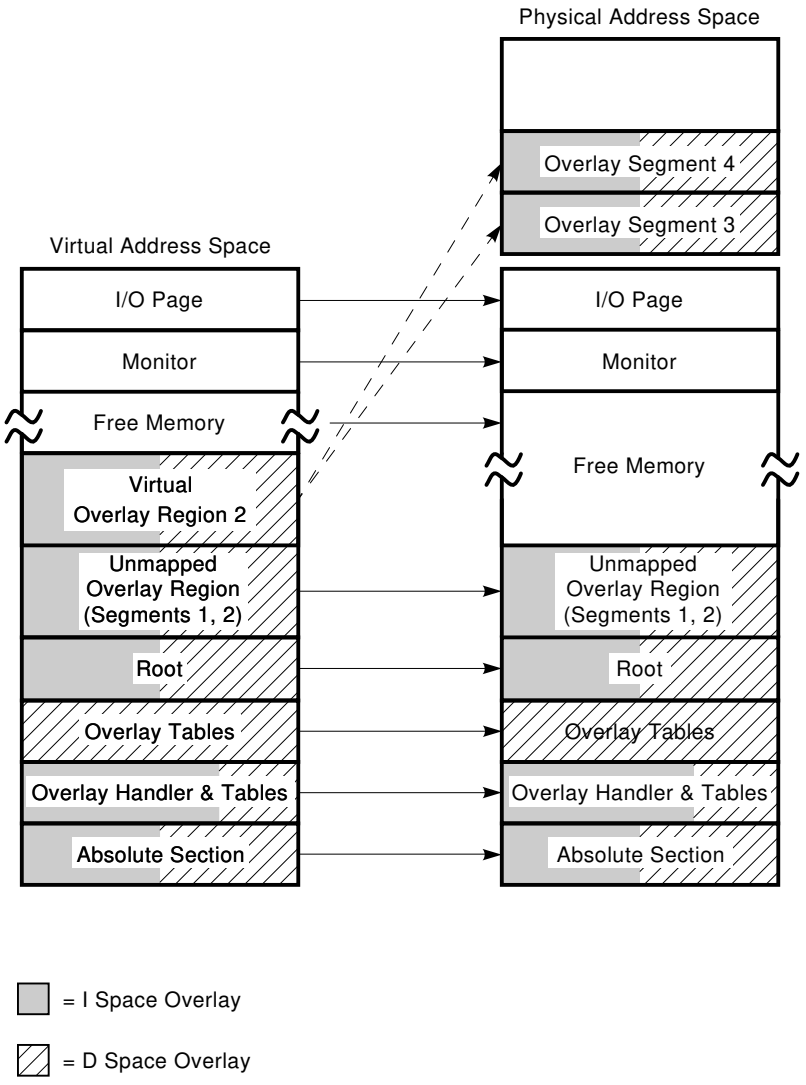
The following example creates an I and D space program with a low-memory overlay region and a virtual overlay region above it.

```
.R LINK
*PROG=PROG/J//
*SEG1/O:1
*SEG2/O:1
*SEG3/V:2
*SEG4/V:2//
```

Figure A–9 shows how this job would be mapped if the program from this example were loaded. See the *Introduction to RT–11* for more detail on I and D space jobs mapping.

Separate I and D Space Overlays

Figure A–9: Memory Diagram Showing Low-Memory I and D Space Overlays



I and D Space Overlay Handler (ZHANDL)

```
.TITLE  ZHANDL    - Disk and Mapped "More" Mapping Overlay Handler
OVR$DK=1
OVR$MP=1
OVR$ID=1
OVR$XH=0
OVR$LO=1

.MCALL  .MODULE
.MODULE UHANDL,VERSION=16,COMMENT=<Universal Overlay Handler>, ....

;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; the information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.

.SBTTL  Conditional Summary
.NLIST  CND
;+
;COND
;
;       OVR$DK  0       No /O overlays
;               (1)     Support /O overlays
;       OVR$MP  0       No /V overlays
;               (1)     Support /V overlays
;       OVR$ID  0       Support I=D environment
;               (1)     Support I<>D & Supy environments
;       OVR$XH (0)      Support multiple overlays
;               1       Support just 1 overlay ((X|Y)HANDL)
;       OVR$LO  0       Overlay handler inits /O area
;               1       Loader inits /O area
;              (OVR$ID) Defaults to same as I&D support
;
;                      OVR$DK or OVR$MP must be on (or both)
;                      OVR$XH forces not OVR$DK
;                      OVR$XH forces OVR$MP
;                      OVR$ID forces OVR$MP
;-
; MAS,SHD,LCP,DBB,JFW,DBB,JFW

OVR$MP=1

.ASECT
.=32                    ;second word of EMT vector in IMAGE
.BYTE   OVR$DK+<2*OVR$MP>+<4*OVR$ID>+<10*OVR$XH>+<20*OVR$LO>
.BYTE   .UHAND

.OHAND  ==   .UHAND

.SBTTL  The Run-Time Overlay Handler
.DSABL  GBL
```

```
;+
; The following code is included in the user's program by the
; linker whenever low memory overlays are requested by the user.
; The run-time low memory overlay handler is called by a dummy
; subroutine of the following form:
;
;       .PSECT $ZTABL,I,GBL,OVR
;       JSR    R5,$OVRH     ;Call to common code for low memory overlays
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  <OVERLAY # *14>   ;# of desired segment
;       .WORD  <ENTRY ADDRESS>   ;Actual core address (virtual address)
;
; One dummy routine of the above form is stored in the resident portion
; of the user's program for each entry point to a low memory overlay
; segment. All references to the entry point are modified by the linker
; to be references to the appropriate dummy routine.  Each overlay
; segment is called into core as a unit and must be contiguous in core.
; An overlay segment may have any number of entry points, to the limits
; of core memory. Only one segment at a time may occupy an overlay region.
;
; If overlays in extended memory are specified, the following dummy
; subroutine is used as the entry point to the extended memory overlay
; handler.
;
;       .PSECT $ZTABL,I,GBL,OVR
;       JSR    R5,$OVRHV   ;Call to common code for low memory overlays
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  <OVERLAY # *14>           ;# of desired segment
;       .WORD  <VIRTUAL ENTRY ADDRESS>   ;Virtual address of segment
;
; The segment number is an index into a table of overlay loading blocks.
; The entries for the /O form of overlays contain the following:
;
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  I-Segment start address
;       .WORD  I-File block number
;       .WORD  I-Word count
;       .WORD  D-Segment start address
;       .WORD  D-File block number
;       .WORD  D-Word count
;
; The "I-" indicates I space and "D-" D space.  If there is no D space
; associated with the I space segment, the three D space words are present
; but contain zeros.
;
; The entries for the /V form of overlays contain the following:
;
;       .PSECT $OTABL,D,GBL,OVR
;       .WORD  I-WDB address
;       .WORD  I-File block number
;       .WORD  I-Word count
;       .WORD  D-WDB address
;       .WORD  D-File block number
;       .WORD  D-Word count
;
; The "I-" indicates I space and "D-" D space.  If there is no D space
; associated with the I space segment, the three D space words are present
; but contain zeros.
;
; Additional data structures in the extended memory overlay handler and the
; overlay table permit use of extended memory. One region definition
; block is defined in the handler and XM EMTs are also included.  Window
```

```
; definition blocks for the extended memory partitions follow the dummy
; subroutines in the overlay table.
;
; There is a three word subroutine prefixed to every I space overlay region
; that returns the segment number of the currently resident segment in R0.
; This value is an index into the overlay table and points at the overlay
; segment information.  Since all D space overlay segments are associated
; with a corresponding I space overlay segment, they contain no prefix.
;
; Undefined globals in the overlay handler must be named "$OVDF1" to
; "$OVDFn" such that a range check may be done by LINK to determine if
; the undefined global name is from the overlay handler. A check is
; done on the .RAD50 characters "$OV", and then a range check is done on
; the .RAD50 charaters "DF1" to "DFn". These global symbols do not appear
; on link maps, since their value is not known until after the map has been
; printed.  Currently $OVDF1 to $OVDF6 are in use.
;
; Global symbols O$READ and O$DONE are useful when debugging overlaid
; programs.
;
;       O$READ will appear in the LINK map and locates the .READW
;       statement in the overlay handler.
;
;       O$DONE will appear in the LINK map and locates the first
;       instruction after the .READW in the overlay handler.
;
;-

.MCALL  .READW    ..V1..
.MCALL  .WDBDF    .RDBDF      .PRINT      .EXIT   .CRAW
.MCALL  .CKXX     .ASSUME     .BR
        ..V1..                ;V1 format
        .WDBDF                ;Define WDB offsets
        .RDBDF                ;Define RDB offsets

        .CKXX   <R0,R1,R1A,R2,R2A,R5,R5A>
C.WDB=1234                    ;check value for WDB address
C.OVR=2234                    ;check value for overlay address
C.ONUM=60                     ;check value for overlay number

.LIBRARY "SRC:SYSTEM"
.MCALL  .EMTDF    .ERRDF      ..READ
.MCALL  .OTBDF    .OTJDF      .OVRDF     .SYCDF   .UEBDF    .VTBDF
.MCALL  ..CRAW
        ..CRAW      ;.CRAW request offsets and values
        ..READ
        .EMTDF      ;define EMT request numbers
        .ERRDF      ;define error numbers
        .OTBDF      ;overlay table entry definitions (/O)
        .OTJDF      ;overlay jump entry
        .OVRDF      ;overlay handle -- SIPP communications
        .SYCDF      ;SYSCOM area
        .UEBDF      ;user error bit masks
        .VTBDF      ;overlay table entry definitions (/V)

OVRCHN  =:  17      ;overlay channel number
```

I and D Space Overlay Handler (ZHANDL)

```
.GLOBL  $OVDF1
.GLOBL  $OVDF2
.GLOBL  $OVDF3
.GLOBL  $OVDF4
.GLOBL  $OVDF5
.GLOBL  $OVDF6
.WEAK   $OVTAB
.WEAK   O$READ
.WEAK   O$DONE
.WEAK   $ODF1
.WEAK   $ODF2
.WEAK   $ODF4
.WEAK   $ODF5

.SBTTL  OVERLAY HANDLER CODE

.PSECT  $OHAND,I,GBL,CON
.PSECT  $ODATA,D,GBL,OVR
.PSECT  $OTABL,D,GBL,OVR
.PSECT  $ZTABL,I,GBL,OVR
.PSECT  $OHAND

.ENABL  LSB

$OVRHZ::              ;Global used to load this variant from library
        .WEAK   $OVRHV
        .WEAK   $OVRH
                                CK.R5=OTJ.JS+4
;+
; There are two entry points to the overlay handler: $OVRHV for /V
; (extended memory) overlays, and $OVRH for /O (low memory) overlays.
;-

$OVRHV::                        ;/V overlay entry point
        INCB    VFLAG           ;Set /V overlay entry switch
$OVRH::    ;/O overlay entry point
                                CK.R5A=OTJ.JS+4
        MOV     R0,-(SP)        ;Save registers
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        ADD     #$OVDF6-$OVJSR-4,R5   ;Find "inline" parameters
                                      ;-4 represents length of JSR R5,X
        MOV     @R5,R1                ;Pickup overlay number
        ADD     #$OVTAB-<OTB.ES*2>,R1 ;Calculate table address
                                      ;Adjusting for index value
                                CK.R1=VTB.WD
                                CK.R1A=OTB.AD
                                CK.R1 VTB.WD,+2
                                CK.R1A OTB.AD,+2
        MOV     (R1)+,R2              ;Get first arg. of overlay seg. entry
                                CK.R2=C.WDB
                                CK.R2A=C.OVR
        TSTB    VFLAG                 ;Is this /V entry?
        BEQ     60$                   ;If non-zero then no
```

```
;+
; Virtual overlay segments in the same region but in different partitions
; use different WDBs.  Only one of these windows exists at any time.
; This is because when a new window in a virtual overlay region is created,
; the monitor implicitly eliminates any window that exists in that
; virtual overlay region.  Thus, if the called overlay segment is not
; currently mapped, its window must be re-created (.CRAWed) besides
; being mapped.  The mapping is done implicitly in the following code
; since the WS.MAP bit is set in all of the virtual overlay segments'
; WDBs.
;-
                                CK.R2 C.WDB
                                .Assume W.NID EQ 0
        TSTB    @R2     ;Do we need to create a window (.CRAW)?
        BEQ     30$     ;Yes
                                CK.R2 C.WDB
        CLR     R0
        CALL    @W.NBAS(R2)     ;Get index of segment now mapped
                                CK.R0=C.ONUM
        BEQ     30$             ;There isn't one; we must .CRAW
                                CK.R0 C.ONUM
                                .Assume VTB.WD EQ 0
                                CK.R2 C.WDB
        CMP     $OVTAB-<OTB.ES*2>(R0),R2  ;Is overlay region same as this one?
        BEQ     50$                       ;If equal, just worry about disk I/O
30$:
        MOV     R2,AREA+A.WDB           ;Set pointer to WDB for .CRAW request
        CALL    O$CRAW                  ;Do the EMT for .CRAW
                                CK.R1 VTB.BK
                                CK.R1A OTB.BK
        MOV     VTB.ES-VTB.BK(R1),AREA+A.WDB ;Point to D-space WDB (if any)
        BEQ     50$                     ;Branch if no D-space WDB
        CALL    O$CRAW                  ;Do the EMT for .CRAW
50$:
                                CK.R2 C.WDB
        MOV     W.NBAS(R2),R2           ;Get memory address of overlay
                                CK.R2=C.OVR
60$:
                                CK.R2A C.OVR ;@R2 is first word in overlay
        CLR     R0      ;Return 0 if no overlay present
        CALL    @R2     ;Ask for overlay number (returned in R0)
                                CK.R5 OTJ.OV,+2
                                CK.R5A OTJ.OV,+2
        CMP     (R5)+,R0     ;Is overlay already resident?
        BEQ     80$          ;Yes, branch to it
        MOV     #<..ISPA!..CURR!..EMIO>,R3 ;Set current mode / I space
        CALL    O$READ       ;Do a mapping-specified read
        MOV     (R1)+,R2     ;Get address of D-space segment or WDB
        BEQ     80$          ;Branch if there isn't a D-space segment
        TSTB    VFLAG        ;Is this /V entry?
        BEQ     70$          ;If zero then no
        MOV     W.NBAS(R2),R2          ;Get address of D space segment from WDB
70$:    MOV #<..DSPA!..CURR!..EMIO>,R3 ;Set current mode / D space
        CALL    O$READ                 ;Do a mapping-specified read
80$:                                   ;Restore users registers
        CLRB    VFLAG                  ;Clear /V flag
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
                                CK.R5 OTJ.AD
                                CK.R5A OTJ.AD
```

```
        MOV     @R5,R5              ;Get entry address
        RTS     R5      ;Enter overlay routine and restore user's R5

90$:
;+
;ERROR
        EMT     ...ERR              ;System error 10 (OVERLAY I/O)
        .BYTE   0,ER.OVE
;This get converted by the monitor into SERR -5 "Error reading an overlay"
;-
O$CRAW:
        .CRAW   #AREA,CODE=NOSET;Do the EMT for .CRAW
        BCS     100$  ;Carry set means error!
        RETURN

O$READ::
        MOV     #RAREA+A.BLK,R0      ;Point to request area
                                CK.R0=RAREA+A.BLK
                                CK.R1 VTB.BK,+2
                                CK.R1A OTB.BK,+2
                                CK.R0 RAREA+A.BLK,+2
        MOV     (R1)+,(R0)+          ;Set block number
;>>>                CK.R2
                                CK.R0 RAREA+A.BUF,+2
        MOV     R2,(R0)+             ;Set buffer address
                                CK.R1 VTB.SZ,+2
                                CK.R1A OTB.SZ,+2
                                CK.R0 RAREA+A.WCNT,+2
        MOV     (R1)+,(R0)+          ;Set word count
                                CK.R0 RAREA+A.TYPE,+2
        MOV     R3,@R0               ;Set current mode / I space
        MOV     #RAREA,R0    ;Point to beginning of request again
                                CK.R0=RAREA
                                CK.R0 RAREA
        EMT     ...REA              ;Issue a mapping read request
O$DONE::BCS     90$    ;Error (stack alignment NOT required)
        RETURN

; ERROR MESSAGE

100$:   MOV        #MSG2,R0        ;Otherwise error
        .PRINT                     ;And print message
        BISB    #<FATAL$>,@#$USRRB ;Set error byte
        .EXIT                      ;And exit

.DSABL  LSB
        .PSECT  $ODATA

.ENABL  LC
.NLIST BEX
;+
;ERROR
MSG2:   .ASCIZ     /?ZHANDL-F-Window error/
;-
        .EVEN

.LIST BEX
RAREA:  .BYTE     OVRCHN,.READ
        .BLKW   <L.REAU-4>/2
        .WORD   ..WTIO

VFLAG:  .BYTE     0        ;/V flag, initially zero
OFLAG:  .BYTE     1        ;One-time flag, initially one
```

```
AREA:   .WORD      .CRAW*^O400+..CRAW,$OVDF4 ;EMT area block for .CRAW
                              .Assume .-AREA GE L.CRAW

$ODF5:: .WORD   $OVDF5       ;Pointer to word after WDBs in overlay table
$ODF4:: .WORD   $OVDF4       ;Pointer to start of WDBs in overlay table

RGADR:  .WORD     0          ;Three word region definition block
RGSIZ:  .WORD     $OVDF3,0   ;$OVDF3 set by LINK = size of region
                              .Assume RGADR+R.GSIZ EQ RGSIZ


$ODF1:: .WORD    $OVDF1      ;High addr root segment + 2 (nxt avail)
$ODF2:: .WORD    $OVDF2      ;High addr /O overlays + 2 (nxt avail)
OVTAB:                       ;The following are required by SIPP
                              .Assume $ODF5 EQ OVTAB+OVR.EW
                              .Assume $ODF4 EQ OVTAB+OVR.SW
                              .Assume RGSIZ EQ OVTAB+OVR.RS
                              .Assume $ODF1 EQ OVTAB+OVR.ER
                              .Assume $ODF2 EQ OVTAB+OVR.EO

.SBTTL  $OVTAB    OVERLAY TABLE

;+
; OVERLAY TABLE STRUCTURE:
;
; LOC 64 ->  $OVTAB:
;            .WORD  <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O overlays
; LOC 66 ->  .WORD  <WDB ADDR>,<RELATIVE BLK>,<WORD COUNT>    /V overlays
;            Dummy subroutines for all overlay segments
; $ODF4 ->   Window definition blocks for extended memory overlays (/V)
; $ODF5 ->   Word after the end of the window definition blocks (/V)
;
;NOTE: description incomplete
;-

.PSECT  $OTABL
$OVTAB::                          ;first argument block if I-D version
.PSECT  $ZTABL
$OVJSR::                          ;first JSR in table if I-D version
.END
```

Index

A

Absolute section
  description, 15–15
.ASECT
  See Absolute section

B

Backup utility program
  See BUP
Binary comparison program
  See BINCOM
BINCOM
  byte-by-byte comparison, 2–4
  command syntax, 2–2
  DCL equivalents, 2–8
  device comparison, 2–4
  differences file
    forcing creation of, 2–4
  differences output format, 2–5
  examples, 2–6
  function of, 2–1
  help option, 2–4
  options (table), 2–4
  output, 2–1, 2–5
  SIPP command file as output from, 2–6, 2–7
    forcing creation of, 2–4
  specifying end of comparison for, 2–4
  specifying starting block for, 2–4
  suppressing differences output, 2–4
  using wildcards with, 2–3
    in both file specifications, 2–3
    in one file specification, 2–3
BUP
  backing up files, 3–15
  backing up logical disks, 3–17
  backing up to magtapes, 3–21
  backing up volumes, 3–16
  calling, 3–2
  command-line syntax, 3–3
  DCL equivalents, 3–36
BUP (Cont.)
  default operation, 3–5
  directory operation, 3–5
  features, 3–1
  image-mode copy, 3–5
    example, 3–16
    for files, 3–15
    for volumes, 3–16
  initializing backup volumes for, 3–6
  initializing volumes for use with, 3–11
  listing saveset and subset directories, 3–22
  options
    alphabetical summary, 3–5
  restore operation, 3–6, 3–28
  summary of operations, 3–7
  terminating, 3–2
  using wildcards with, 3–4
  verifying a data transfer, 3–13

C

CCL (Concise Command Language), 1–10
  command-line syntax, 1–10
Comparison
  binary files
    See BINCOM
CONFIG
  command-line syntax, 4–1
  definition, 4–1
  examples, 4–4
  options, 4–2
Configuration utility program
  See CONFIG
CONSOL
  changing system, 4–5
  definition, 4–5
Console utility program
  See CONSOL
CSI (Command String Interpreter), 1–8
  command-line syntax, 1–8

D

DATIME
  definition, 4–6
  running, 4–6
  two versions, 4–6
Datime utility program
  See DATIME
DCL equivalents
  LIBR
  BINCOM, 2–8
  BUP, 3–36
  DIR, 5–13
  DUMP, 6–12
  DUP, 7–34
  ERROUT, 9–15
  FILEX, 10–17
  FORMAT, 11–9
  LD, 12–2
  LINK, 15–45
Device comparison
  binary
    See BINCOM
Device Utility Program
  See DUP
Differences between binary files
  See BINCOM
DIR, 5–1
  calling, 5–1
  command-line syntax, 5–2
  DCL equivalents, 5–13
  options
    descriptions, 5–3
    summary, 5–11
  terminating, 5–1
Directory listings
  default format, 5–6
  excluding certain files from, 5–7
  of deleted files, 5–7
  of files' starting block numbers, 5–3
  of files created before a specified date, 5–5
  of files created since a specified date, 5–5
  of files with a specified date, 5–4
  of only file names and types, 5–4
  of protected files, 5–10
  of unprotected files, 5–10
  of unused areas, 5–4, 5–6
  reading, 5–1
  sorting by
Directory listings
  sorting by (Cont.)
    alphabetical order, 5–3
    creation date, 5–8
    file type, 5–8
    position on volume, 5–8
    reverse order, 5–8
    size, 5–8
  specifying number of columns for, 5–3
  specifying sorting for, 5–8
  starting with file you specify, 5–5
  summary format, 5–6
  with octal sizes and block numbers, 5–7
  with volume ID and owner name, 5–10
DIRECTORY utility program
  See DIR
DUMP
  calling, 6–1
  command syntax, 6–2
  DCL equivalents, 6–12
  examples, 6–8, 6–9, 6–10
  halting, 6–1
  operations with magtape, 6–4
  options (table), 6–3
DUMP utility program
  See DUMP
DUP
  algorithm for directory segments, 7–27
  calling, 7–1
  command-line syntax, 7–2
  DCL equivalents, 7–34
  options
    bad-block scan, 7–13
    boot, 7–15
    bootstrap-copy, 7–20
    changing default directory size, 7–27
    changing volume ID and owner, 7–29
    combinations, 7–2
    covering bad blocks, 7–32
    create, 7–5
    directory-initialization, 7–25
    extend, 7–19
    file, 7–7
    image-mode copy, 7–9
    no-query, 7–24
    replacing bad blocks, 7–30
    restoring a disk, 7–33
    squeeze, 7–17
    summary, 7–3

DUP
options (Cont.)
types
action, 7–2
mode, 7–2
volume ID, 7–22
wait-for-volume, 7–23
terminating, 7–1

E

EDIT
calling, 8–1
command-line syntax, 8–7
commands
character-oriented, 8–8
close, 8–15
descriptions, 8–15
input, 8–19
line-oriented, 8–9
open, 8–15
output, 8–19
pointer-relocation, 8–24
repeating, 8–10
rules for entering, 8–12
search, 8–26
summary, 8–41
text listing, 8–29
text modification, 8–31
types, 8–14
utility, 8–36
error conditions, 8–43
key commands, 8–5
location pointer, 8–2
memory usage, 8–3
modes
command, 8–4
text, 8–4
pages, 8–2
running, 8–2
sample editing session, 8–44
EL.SYS or ELX.SYS, 9–6
ELINIT, 9–6
ERRLOG.REL, 9–6
Error Logger
forms of, 9–1
using with a multi-job monitor, 9–11
using with a single-job monitor, 9–9
Error logging
components, 9–4
Error logging (Cont.)
definition, 9–1
diagrams, 9–8
enabling, 9–12
functions, 9–2
programs, 9–6
Error-Logging Package
See Error logging
Error-log reports
displaying, 9–13
MSCP
example, 9–24
types, 9–23
non-MSCP
device, 9–16
examples, 9–20
memory, 9–18
options, 9–14
printing, 9–13
saving, 9–13
ERROUT, 9–7
DCL equivalents, 9–15

F

File Exchange Utility
See FILEX
Files
backing up with BUP, 3–15
listing
See DIR
FILEX
and
DECsystem–10, 10–10
DOS/BATCH, 10–8
interchange diskette, 10–11
RSTS, 10–8
DCL equivalents, 10–17
defaults and wildcards, 10–1
deleting files, 10–6
file formats, 10–2
initializing directories, 10–16
listing directories, 10–7
operating systems, 10–2
option summary, 10–4
option types, 10–3
pausing, 10–15
supported devices, 10–1
FORMAT, 11–1
at nonstandard addresses, 11–2

FORMAT (Cont.)
calling, 11–2
command-line syntax, 11–3
DCL equivalents, 11–9
devices formatted, 11–1
extended device units, 11–2
option descriptions, 11–5
option summary, 11–8
terminating, 11–2
uses, 11–1
Format Utility
See FORMAT

G

Global symbol
creating, 15–20
definition, 15–1
resolving, 15–20

L

LD
calling, 12–1
command-line syntax, 12–2
DCL equivalents, 12–2
device handler and utility, 12–1
option descriptions, 12–3
option summary, 12–2
terminating, 12–1
uses, 12–1
LET
defining keys for substitution, 13–2
defining symbols for substitution, 13–1
definition, 13–1
definitions in STRTxx.COM file, 13–3
deleting substitutions, 13–2
enabling, 13–1
options, 13–2
LET Substitution Utility
See LET
LIBR
calling, 14–2
command-line syntax, 14–2
DCL equivalents, 14–16
library
directory, 14–6
macro
creating, 14–13
options, 14–13
LIBR
library (Cont.)
merging, 14–5
object
creating, 14–4
storing, 14–4
referencing, 14–3
storing in, 14–3
options
combining, 14–12
descriptions, 14–7
terminating, 14–2
uses, 14–1
Librarian Utility
See LIBR, 14–1
Library file
definition, 14–1
Library module
definition, 15–7
description, 15–7
multiple-definition, 15–9
processing of, 15–7
LINK
calling, 15–2
command-line syntax, 15–3
default devices, 15–3
definition, 15–1
functions
descriptions, 15–1
order, 15–2
option
DCL equivalents, 15–45
descriptions, 15–21
summary, 15–42
overlays
combining low- and extended-memory, A–28
creating, A–1
extended-memory, A–14
extended-memory handler, A–22
extended-memory load map, A–19
guidelines for creating, A–6, A–39
I and D space, A–39
I and D space handler, A–41
low-memory, A–2
low-memory overlay handler, A–10
one segment, A–30
pseudo handler, A–34
prompts, 15–4
terminating, 15–2

Linker Utility
See LINK
Load map
description, 15–13
Load module
definition, 15–1
description, 15–11
structure, 15–15
Local symbol
definition, 15–1
Logical disks
See LD
backing up, 3–17
backing up files into, 3–20
directory, 3–27
Logical-Disk Subsetting Utility
See LD

M

Magtapes
dumping, 6–4

O

Object module
definition, 15–1
description, 15–6
OBJ file, 14–1
OHANDL, A–10
Output module
See Load module or Load map
Overlays
See LINK

P

Program section
description, 15–15
.PSECT
See Program section

R

Restoring BUP volumes and files, 3–6

S

Saveset
definition, 3–1
Single-Line Text Editor
Single-Line Text Editor (Cont.)
See EDIT
SIPP
input command file
creating with BINCOM, 2–4, 2–6, 2–7
SML file, 14–1
Subset
definition, 3–7
Substitution
symbols for character strings, 13–1
symbols for keys, 13–2

T

Transferring files
FILEX
File Exchange Utility, 10–1

U

Unsupported utilities, 1–1
Utilities
definition, 1–1
summary description, 1–2, 1–4
types, 1–6
unsupported, 1–1

V

VHANDL, A–22
Volumes
backing up with BUP, 3–16

W

Wildcards
BUP treatment of, 3–4

X

XHANDL, A–34

Z

ZHANDL, A–41