digital

VAX–11
Utilities Reference Manual

Order No. AA-H781A-TE

VAX11

March 1980

This document describes utility programs for use on VAX-11 processors.

VAX-11
Utilities Reference Manual

Order No. AA-H781A-TE

SUPERSESSION/UPDATE INFORMATION: This is a new document for this release. Chapter 3 supersedes and replaces Chapter 3 and Appendixes C and D of the VAX-11 Text Editing Reference Manual. Chapter 4 supersedes and replaces the VAX-11 Disk Save and Compress User's Guide

OPERATING SYSTEM AND VERSION: VAX/VMS V02

SOFTWARE VERSION: VAX/VMS V02

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, March 1980

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

FIGURES

TABLES

PREFACE

This reference manual describes utility programs supported by DIGITAL on the VAX/VMS operating system.

INTENDED AUDIENCE

This manual is intended for users who are already familiar with VAX/VMS system concepts. Use of the various utility programs is appropriate for users at different levels of experience and responsibility. The expected user group for each program is defined below in the chapter summaries.

STRUCTURE OF THIS DOCUMENT

This manual is organized into eight chapters and one appendix.

Each chapter of this manual describes one utility program, except for Chapter 3, which includes two related editors (SLP and SUMSLP), Chapter 4 includes three variants of a disk back-up program (DSC) and Chapter 6 describes two variants of a verification utility. Each chapter contains a list of the messages issued by the utility. The following are the contents and intended audience of each chapter.

Chapter 1 describes the Personal Mail Utility, referred to as MAIL. This program allows users to send messages to one another, within the same system or between any VAX-11 computers that are connected by means of DECnet-VAX. Use of MAIL is appropriate for all system users.

Chapter 2 describes the File Transfer Utility, referred to as FLX (and generally pronounced 'FILEX'). This program transfers files from one volume to another and performs volume format conversions. FLX is intended for use by all system users.

Chapter 3 describes two related batch-oriented text editors, SLP and SUMSLP. These editors are used to incorporate changes into source files and to indicate these changes with an audit trail. SLP and SUMSLP are intended for use by all system users.

Chapter 4 describes the Disk Save and Compress Utilities, referred to as DSC1, DSC2, and DSC-2 (stand-alone). The DSC programs are used to back up and restore disk volumes that have been formatted and initialized as Files-11 volumes. These programs are intended for VAX/VMS system managers, operators, system programmers, and application programmers.

Chapter 5 describes the Bad Block Locator Utility, referred to as BAD. This program determines and records the number and location of bad blocks on block-structured volumes. BAD is intended for use by VAX/VMS system managers, operators, and system programmers.

Chapter 6 describes the File Structure Verification Utilities, referred to as VFY1 and VFY2 (and pronounced 'VERIFY'). This program checks the readability and validity of Files-11 volumes. It is intended for use by VAX/VMS system managers, operators, and system programmers.

Chapter 7 describes the Librarian Utility, referred to as the Librarian. This program allows you to store useful modules in a central, easily accessible location. It is intended for use by all VAX/VMS users.

Chapter 8 describes the Message Utility. This program allows you to construct your own informational, warning, and error messages, or to customize the messages provided by VAX/VMS. The Message Utility is intended for use by VAX/VMS system programmers and application programmers.

Appendix A lists the Files-11 structured devices supported by VAX/VMS, with the characteristics and device code of each device. It presents information needed by users of several of the utilities in this manual.

ASSOCIATED DOCUMENTS

To use the utilities described in this document, you should be familiar with the following manuals:

- VAX/VMS Primer
- VAX/VMS Command Language User's Guide
- VAX/VMS Summary Description

Some of the utilities require familiarity with disk structures and volume concepts described in the following manuals:

- VAX/VMS System Manager's Guide
- Introduction to VAX-11 Record Management Services
- VAX-11 Record Management Services Reference Manual

Some of the utilities run in compatibility mode; readers may wish to consult the VAX-11/RSX-11M User's Guide.

Note that there are other utility programs that run on VAX-11 processors - PATCH and the System Dump Analyzer, for example. These programs are described elsewhere in the VAX-11 documentation set. For a complete list of VAX-11 documents, including a brief description of each, see the VAX-11 Information Directory and Index.

CONVENTIONS USED IN THIS DOCUMENT

The following conventions are observed in this manual, as in the other VAX-11 documents:

Convention	Meaning
Uppercase words and letters	Uppercase words and letters, used in examples, indicate that you should type the word or letter exactly as shown.
Lowercase words and letters	Lowercase words and letters, used in format examples, indicate that you are to substitute a word or value of your choice.
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.
[ ]	Square brackets indicate that the enclosed item is optional.
{ }	Braces are used to enclose lists from which one element is to be chosen.
...	A horizontal ellipsis indicates that the preceding item(s) can be repeated one or more times.
. . .	A vertical ellipsis indicates that not all of the statements in an example or figure are shown.
RET or <RET>	A symbol with a 1- to 3-character abbreviation indicates that you press a key on the terminal, for example, RET .
CTRL/X or <CTRL/x>	The phrase <CTRL/x> indicates that you must press the key labeled CTRL while you simultaneously press another key, for example <CTRL/C>, <CTRL/Y>, <CTRL/O>. In examples, this control key sequence is shown as ^x, for example ^C, ^Y, ^O, because that is how the system echoes control key sequences.

Unless otherwise noted, all numeric values are represented in decimal notation.

Unless otherwise specified, you terminate commands by pressing the RETURN key.

Where the term file-spec is used in this document, it refers to a file specification constructed according to the following definitions, with the format:

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

The punctuation marks (colons, brackets, period, semicolon) are required syntax that separate the various components of the file specification.

node

A network node name. This is applicable only to systems that support DECnet-VAX.

device

The device on which the file is stored or is to be written.

directory

The name of the directory under which the file is cataloged on the device specified. You can delimit the directory name with square brackets, as shown, or with angle brackets (<>).

filename

The file by a name of up to 9 alphanumeric characters.

type

The type of data in the file; type can be up to 3 alphanumeric characters.

version

The version of the file. Versions are identified by a decimal number, which is increased by 1 each time a new version of the file is created. Either a semicolon or a period can be used to separate type and version.

You need not always state all elements of a file specification explicitly. Frequently, only the file name is required. If you omit other parts of the file specification, a default value is provided as described in the following table.

Optional Element	**Default Value**
node	Local network node
device	User's current default device
directory	User's current default directory
type	Depends on use, as described with the various utilities
version	Input: highest existing version Output: highest existing version plus 1

Any variations in file specifications that a utility program requires are stated in the description of that utility.

CHAPTER 1

PERSONAL MAIL UTILITY

The Personal Mail Utility (MAIL) allows you to send messages to other users on your system or on any other VAX-11 computer that is connected to your system by means of DECnet-VAX. You can also file, forward, delete, print, and reply to messages that other users send to you.

Messages that you receive are stored in files called message files. Your default message file, called MAIL.MAI, is created in your default login directory the first time you receive mail. New messages that you receive are appended to the end of MAIL.MAI. You can copy messages to other message files that you name using the FILE command. Message files are described in detail in Section 1.3. Figure 1-1 shows a sample message file.

```
From:  HDV                    23-JAN-80  16:25
To:    @GROUP
Subj:  Change of Location

Next Tuesday's review meeting has been moved from Conference Room
A to Conference Room D (on the second floor).  See you there.

From:  MENDOZA                24-JAN-80  10:06
To:    READER
Subj:  Payroll program

There is a new version of the payroll program in my directory:
[MENDOZA]PAYROLL.FOR;12.

From:  NOEL                   28-JAN-80  11:46
To:    READER
Subj:  Source statement format

You may include as many spaces or tabs as you want!
```

Figure 1-1 MAIL Message File

1.1 INVOKING MAIL

The DIGITAL Command Language (DCL) command MAIL can be used either to invoke the MAIL utility or, if specified with parameters, to send a file to another user and return you to DCL.

To invoke MAIL, enter the following command in response to the DCL prompt:

```
$ MAIL
```

The utility responds with the prompt:

```
MAIL>
```

You can now issue the commands described in Section 1.2 to send and display messages.

When the MAIL command is used with parameters, the command string has the following format:

```
MAIL[/SUBJECT:"text"] [file-spec] [username[,...]] ["@listname"]
```

text

The subject of the message. If you include more than one word, you must enclose the text in quotation marks.

If you omit the /SUBJECT qualifier, the message is sent without a subject notation.

file-spec

A file containing message text to be sent. If you omit the file type, the default file type is TXT. No wild card characters are allowed in the file specification.

If you do not specify a file in the command string, the MAIL utility is invoked, as shown above.

username[,...]

One or more users to receive the message. Sending mail to multiple users is described in Section 1.2.12.

If you do not specify a user name in the command string, you will be prompted for the user name.

"@listname"

The name of a distribution list file. The default file type is DIS. Setting up a distribution list file is described in Section 1.2.12.2. The quotation marks and at sign are required. A distribution list name that follows a user name specification must be preceded by a comma.

When MAIL sends you a message from another user, and you are logged in, MAIL notifies you with a message on your terminal. For example:

```
New Mail from Weed
```

You will also be notified that you have new mail when you log in and when you invoke the MAIL utility.

1.2 MAIL COMMANDS

MAIL commands consist of one word typed in response to the prompt MAIL>. These commands can be abbreviated to a unique, shorter form (usually as short as one letter). Note that D is the short form of DELETE (not DIRECTORY) and R is the short form of REPLY (not READ).

Table 1-1 summarizes the MAIL commands, which are described in the following sections.

Table 1-1
Summary of MAIL Commands

Command	Meaning
BACK	Backs up to the previous message
DELETE	Deletes the current (last-read) message
DIRECTORY	Lists a summary of your messages
EXIT	Exits from MAIL
FILE	Copies current (last-read) message into a specified file
FORWARD	Forwards current (last-read) message to user or users
HELP	Displays information on how to use MAIL
NEXT	Skips to the next message
PRINT	Prints the current (last-read) message
READ and <RETURN>	Displays next page, next message, or (READ only) specified message
REPLY	Sends a reply to the sender of the current (last-read) messsage
SEND	Sends a message to a user or users

1.2.1 BACK Command

The BACK command displays the message preceding the current (last-read) message.

1.2.2 DELETE Command

The DELETE command deletes the current (last-read) message.

You must be reading a message in order to delete it. For example:

```
MAIL> READ

From:  WEED     19-APR-79 19:15
To:    PIERSON
Subj:  Monthly Meetings
       .
       .
       .
       (message text)
       .
       .
       .

MAIL> DELETE
```

Although this message is apparently deleted when you type DELETE, it is not actually deleted from the file until you exit from MAIL or read messages from another file. Thus, if you accidentally issue the DELETE command to delete a message, you can abort the command by typing <CTRL/Y>.

1.2.3 DIRECTORY Command

The DIRECTORY command lists a summary of the messages in the current or specified message file, including message number, sender's name, date, and subject:

```
DIRECTORY [filename]
```

filename

A message file. You only specify the filename element of the file specification (up to nine alphanumeric characters); file type MAI is assumed.

If a file name is specified, MAIL displays a summary of the messages in the file. If no file name is specified, MAIL displays a summary of the messages in the current file, as follows:

```
MAIL> DIRECTORY

     # From           Date          Subject

     1 MURRY           7-SEP-1979   FIELD TEST
     2 STAR::PETER    24-SEP-1979   Code Review
     3 MARCEL         26-SEP-1979   Monthly Meetings
     4 LDAVIS         26-SEP-1979   PASCAL
     5 JOSH           26-SEP-1979   LINKER
```

1.2.4 EXIT Command

The EXIT command allows you to exit from the MAIL program.

```
MAIL> EXIT

$
```

You can also exit from MAIL by typing <CTRL/Z>.

1.2.5 FILE Command

The FILE command is used to save a copy of the current (last-read) message in a specified message file. The copy is appended to the end of the specified file.

```
FILE filename
```

filename

The message file in which the current message is to be saved. You only specify the filename element of the specification (up to nine alphanumeric characters). If the specified message file does not exist, MAIL creates it in your default login directory, giving it the file type MAI.

You must be reading a message in order to file it.

1.2.6 FORWARD Command

The FORWARD command sends a copy of the current (last-read) message to a user or users. MAIL prompts you for the name of the user or users to whom you want to forward the message. See the SEND command for more information on sending messages.

You must be reading a message in order to forward it.

1.2.7 HELP Command

The HELP command allows you to obtain information about the MAIL program.

To obtain information about all of the MAIL commands, type

```
HELP *
```

To obtain information about individual commands or topics, type HELP followed by the command or topic name:

```
HELP DELETE

DELETE

     Deletes the current (last-read) message from your current
      message file.

MAIL>
```

1.2.8 NEXT Command

The NEXT command skips to the next message and displays it. This command is useful if, while reading through your messages, you encounter a particularly long message that you would like to skip over.

1.2.9 PRINT Command

The PRINT command queues a copy of the current (last-read) message for printing. The file(s) created by the PRINT command are not actually released to the print queue until you exit from MAIL so that multiple messages will be concatenated into one print job. The PRINT command takes an optional qualifier, as follows:

```
PRINT [/QUEUE=queue-name]
```

/QUEUE = queue-name

The device on which a message is to be printed. If the qualifier is not specified, the last queue name specified is used. If an explicit queue name has never been specified, SYS$PRINT is used.

You must be reading a message in order to print it.

1.2.10 READ Command

The READ command displays your messages. It can be issued with or without parameters, in the following formats:

```
READ [filename] [message-number]

message-number

<RETURN>
```

filename

A message file. You specify only the filename element of the file specification (up to nine alphanumeric characters); file type MAI and your default login directory are assumed.

If a file name is specified, MAIL will display messages from that file. If no file name is specified, MAIL will display messages from the current file. The default file when you enter the MAIL utility is MAIL.MAI.

message-number

A number representing the position of a message in a message file. If you specify a number greater than the number of messages in the file, MAIL will display the last message in the file. Therefore, to read the latest message in a file, specify a very large message number.

You can display a message by entering only its message number, without the READ command.

<RETURN>

The RETURN key. Pressing this key is the same as entering the READ command without parameters.

The READ command can be issued without parameters. The first time after invoking the MAIL utility that you issue the READ command without parameters, or press <RETURN>, MAIL displays the first page of your oldest unread message from your MAIL.MAI file. If there are no unread messages, MAIL displays the oldest message in the file. Each time you enter the READ command without parameters, or press <RETURN>, MAIL displays the next page, or the next message if there are no more pages in the current message.

1.2.11 REPLY Command

The REPLY command sends a message to the sender of the current (last-read) message.

```
REPLY[/qualifier] [file-spec]
```

qualifier

One of the qualifiers in Table 1-2. If you specify both qualifiers, the /EDIT qualifier is ignored.

file-spec

A file to be sent as your reply. If no file is specified, you will be prompted for the text of your reply.

You must be reading a message in order to reply to it.

1.2.12 SEND Command

The SEND command sends a message to another user or group of users. (If you simply want to send a file to another user or group of users, you may want to use the extended form of the MAIL command described in Section 1.1 instead of using SEND.) You can include a file specification in the SEND command.

```
SEND[/qualifier] [file-spec]
```

See Table 1-2 for a description of the SEND qualifiers.

MAIL prompts you first for the name of the user or users who will receive the message:

```
To:
```

You reply with the user name(s) or with the file name of a distribution list file (as described in Sections 1.2.12.1 and 1.2.12.2), in the following format:

```
[[nodename::]username,...] [,] [@listname]
```

Then MAIL prompts you for the subject of the mail:

```
Subj:
```

If you specify a file in the SEND command, the text in that file is sent to the users.

If you do not specify a file, MAIL displays:

Enter your message below. Press CTRL/Z when complete.

Type the message that you want to send; then press <CTRL/Z>. Note that once you have typed a line and pressed <RETURN>, there is no way to change it. See Table 1-2 for instructions on invoking a text editor to edit your messages.

Table 1-2
SEND and REPLY Qualifiers

Qualifier	Function
/EDIT	Specifies that a text editor is to be called to edit the message that you are sending. If you have included a file specification in the command, that file will be opened for editing. If you have not specified a file, the editor will be invoked so that you can edit your new message. If the logical name MAIL$EDIT is defined, its equivalence name will be used as the name of a command procedure that will invoke the editor. Note that since the MAIL$EDIT command procedure is executed in the context of a subprocess, the definition of MAIL$EDIT and the command procedure itself must not reference any process logical names defined by the initiating process. If MAIL$EDIT is not defined, the command procedure SYS$SYSTEM:MAILEDIT.COM will be called. This command procedure contains the DCL command EDIT, which invokes the SOS text editor. The command that you use to exit from the editor will complete the SEND or REPLY operation. For this qualifier to work properly, your default command interpreter must be DCL.
/LAST	Specifies that the last message that you sent should be used as the text for the message. The /EDIT qualifier is ignored if /LAST is specified.

1.2.12.1 **Sending Messages via DECnet-VAX** - If you include a node name with the user name, the message is sent by means of DECnet-VAX to that user. If you do not specify a node name, MAIL assumes that the user is on your node. If the version of DECnet being used does not provide automatic routing and the node is not connected directly to your node you can specify routing as follows:

nodename1::nodename2::...nodenameN-1::nodenameN::username

nodename1::

The node directly connected to your node.

nodename2::...nodenameN-1

The intermediate nodes through which the message is to be routed.

nodenameN::

The node at which the user who is to receive the message is located.

username

The name of the user who is to receive the message.

You can specify node names and user names as logical names. They are translated like VAX-11 RMS specifications: a node name or user name is only translated if it is the first string in the specification. Any access control information in the node name or logical name is ignored.

NOTE

See the DECnet-VAX System Manager's Guide for information on defining the MAIL network object type.

1.2.12.2 **Sending Messages to Distribution Lists** - If you frequently send mail to the same group of users, you may find it helpful to use a distribution list. A distribution list is a file containing the names of users to whom you want to send messages.

To set up a distribution list, use the DCL command EDIT or CREATE to create a distribution list file with the file type DIS. Enter one user name per line in this file. A distribution list can also include the names of other distribution lists; the depth to which you can nest distribution lists is determined by your Open File Quota. You can include comments by entering lines whose first character is an exclamation point (!). For example:

```
$ CREATE LIS.DIS
!SOFTWARE WRITERS:
PIERSON
NODE3::JOSEPHS
LAWRENCE
NODE4::ASHLEY
```

To use the distribution list file, you enter its file name preceded by an at sign (@) in response to the To: prompt. For example

To: @LIS

You can enter separate user names along with the distribution list if the distribution list is the last entry. For example:

To: GEORGE, MARCEL, BEN, ERICA, @LIS

1.3 MESSAGE FILES

Mail messages are stored in your default login directory as ASCII text, one line per record, in standard variable length record files. Each message is delimited by two records, the first containing only the ASCII form-feed character, and the second beginning with the word "From:". Each line of text can contain a maximum of 255 characters.

The default message file entered when MAIL is invoked is MAIL.MAI.

1.4 SYSTEM MANAGEMENT AND MAIL

A count of the number of new messages that a user has received is stored in the user's system authorization record in SYS$SYSTEM:SYSUAF.DAT. This count is used to make up the message that you receive upon logging in if you have new mail messages waiting to be read. If the user authorization file is replaced with a copy (for example, a backup copy), the count in the file, and therefore the message received upon logging in, may not correspond to the actual mail messages in user's mail files. This inconsistency disappears, however, the first time a message file is read.

Mail keeps SYSUAF.DAT open while MAIL is being run, sometimes preventing this file from being copied.

1.5 MAIL STATUS MESSAGES

This section lists, in alphabetical order, the common status messages you can receive from MAIL. These are in the form:

```
%MAIL-L-message, message-text
```

The L is a severity code, either E for error or W for warning. The message is a mnenomic representing the specific error that occurred. The message-text is a brief description of the condition that caused the message to be issued.

After each message is an explanation of the probable cause of the message and suggested user response. (For more information on messages, refer to Chapter 8.)

%MAIL-E-LOGLINK, network error creating link to node node-name::

> **Explanation:** An error occurred when MAIL attempted to use DECnet-VAX to communicate with another system. This message is usually accompanied by a system error message indicating the reason for the error.
>
> **User Action:** Resolve the problem and reenter the command.

%MAIL-E-NOMOREMSG, no more messages

> **Explanation:** You have no more messages to read.
>
> **User Action:** If you want to read your messages again, enter the READ command; otherwise enter another command in response to the MAIL> prompt.

%MAIL-E-NOSUCHUSR, no such user user-name

Explanation: You attempted to send a message to a user that does not exist as an authorized user of the computer system.

User Action: Reenter the command and specify a valid user name.

%MAIL-W-NOTREADIN, you aren't reading a message

Explanation: The command you typed, (DELETE, FILE, FORWARD, PRINT, or REPLY) is appropriate only when you are reading a message.

User Action: Read the desired message using the READ command and then type the appropriate command in response to the MAIL> prompt.

%MAIL-W-SYNTAX, error parsing

Explanation: You typed an incorrect command or user name.

User Action: Enter the correct command or user name in response to the MAIL> prompt.

%MAIL-E-SENDERR, error sending to user user-name

Explanation: An error occurred when you attempted to send mail to the specified user. This message is usually accompanied by a system error message indicating the reason for the failure.

User Action: Resolve the problem and reenter the command.

%MAIL-E-CREPRIJOB, error creating print job on queue queue-name

Explanation: An error occurred when you attempted to create a print job to queue a message for printing. This message is usually accompanied by a system error message indicating the reason for the failure.

User Action: Resolve the problem and reenter the command.

CHAPTER 2

FILE TRANSFER UTILITY

The File Transfer Utility (FLX) is a utility program that transfers files from one volume to another. FLX can be used on DOS-11, RT-11, and Files-11 formatted volumes. It converts the format of the files, as appropriate, when transferring files between volumes with different formats. For example, when transferring DOS-11 files to Files-11 volumes, FLX converts the DOS-11 files to Files-11 format.

FLX performs file transfers and format conversions from:

- DOS-11 to DOS-11 volumes
- Files-11 to Files-11 volumes
- RT-11 to RT-11 volumes
- DOS-11 to Files-11 volumes
- Files-11 to DOS-11 volumes
- Files-11 to RT-11 volumes
- RT-11 to Files-11 volumes

FLX cannot transfer files directly between DOS-11 and RT-11 volumes.

In addition to transferring files, FLX allows you to:

- Initialize DOS-11 or RT-11 volumes
- List directories of DOS-11 or RT-11 volumes
- Delete files from RT-11 files-structured volumes

FLX recognizes all Files-11 volumes on VAX/VMS devices. It recognizes DOS-11 formatted volumes on the following devices:

TE16, TU45, or TU77 magnetic tape

FLX recognizes RT-11 formatted volumes on the following devices:

TU58 DECtape II data cartridge
RL02 cartridge disk
RK06 or RK07 cartridge disk
RX01 flexible disk
RX02 flexible disk

Files-11 volumes are the default volumes initialized by the DCL command INITIALIZE. They are either File-11 Structure Level 1 or Files-11 Structure Level 2 volumes. DOS-11 and RT-11 volumes are initialized using FLX commands. Since the formats of these volumes are not recognized by VAX/VMS, the volumes must be mounted foreign, that is, by use of the /FOREIGN qualifier. See the VAX/VMS Command Language User's Guide for more information on the INITIALIZE and MOUNT commands.

You can use FLX interactively or through a command procedure. FLX allows only one level of indirect command file specification.

2.1 INVOKING AND TERMINATING FLX

To invoke FLX, enter the following in response to the DIGITAL Command Language (DCL) prompt:

```
$ RUN SYS$SYSTEM:FLX
```

The utility responds with the prompt:

```
FLX>
```

You can now enter any FLX command string. To return to DCL at any time, type <CTRL/Z>.

2.2 FLX COMMAND STRING

Formats for specifying FLX functions vary, as described in Sections 2.4 through 2.6. The following are possible formats for a FLX command; each element in the command string is explained below.

```
device-spec/qualifier
device-spec=file-spec/qualifier
file-spec/qualifier
/qualifier
```

device-spec

The device name and directory for the FLX output device. It takes the form:

```
devu:[directory]
```

The device name (dev) can be any of the 2-character device codes listed below.

DOS-11

Device Code	Device
MT	TE16, TU45, TU77
MS	TS-11

Rt-11

Device Code	Device
DD	TU58
DL	RL02
DM	RK06, RK07
DY	RX02
CS	RX01 (VAX-11/780 console floppy)

Files-11

Device codes for Files-11 devices are listed in Appendix A.

The u is the unit number of the device. FLX does not recognize alphabetic controller designators. You must convert them to RSX unit numbers when specifying devices to FLX; MTA1 must be specified as MT1. The controller designator can still be used, however, in ALLOCATE and MOUNT commands referring to volumes to be used with FLX.[1]

The colon (:) acts as the device name terminator and must follow the device code.

The directory field is optional. The directory specification is subject to restrictions depending on the medium to which it applies.

RT-11 volumes accept no directory specification at all.

DOS-11 volumes accept only directories in the user identification code (UIC) format, for example [310,22]. The two numbers are octal, and must be in the range 0 through 377. When the directory is specified in an input file specification, either number or both may be indicated by a wild card character. If you do not specify a directory, FLX uses your current default directory, if it is in UIC format; otherwise, it uses your process's UIC.

Files-11 volumes accept the standard form of VAX/VMS directory specification documented in the VAX/VMS Command Language User's Guide. Wild card characters may be specified only with a single level of directory or with the UIC format. If you do not specify a directory, FLX uses your current default directory.

file-spec

The file specification for an input or output file.

Wild card characters are valid only for input file specifications. Version numbers are valid only for Files-11 files and cannot be specified as wild card characters. FLX does not accept logical names in file specifications.

FLX does not permit output file specifications. The output files take the names of the input files.

RT-11 volumes use 6-character file names, plus 3-character file types; file names are truncated to six characters when files are copied into RT-11 volumes.

/qualifier

Any of the qualifiers described in Section 2.3. Multiple qualifiers can be used; their order is not important.

---

1. For information on converting VAX/VMS native mode unit numbers to compatibility mode unit numbers, see the explanation of mapping physical device names in the VAX-11/RSX-11M User's Guide.

2.3 FLX QUALIFIERS

FLX uses three types of qualifiers:

- Volume format qualifiers, which specify the format of the volume on which files are stored: Files-11, DOS-11, or RT-11 volumes.
- Transfer mode qualifiers, which specify the format of a file on a non-Files-11 volume. Files can be in formatted ASCII, formatted binary, or file image format.
- Control qualifiers, which provide control functions for use in FLX operations. These qualifiers can be used to specify such items as the number of blocks to be allocated to an output file or the density of a magnetic tape.

These three types of qualifiers are described in detail in the next three sections.

2.3.1 Volume Format Qualifiers

The three volume format qualifiers are used in command strings to define the format of volumes. They can also be used by themselves after the FLX> prompt to change the default for input and output volumes. Table 2-1 describes these qualifiers.

Table 2-1
FLX Volume Format Qualifiers

Qualifier	Function
/DO	Identifies the volume as a DOS-11 formatted volume
/RS	Identifies the volume as a Files-11 formatted volume
/RT	Identifies the volume as an RT-11 formatted volume

Initially, input volumes default to DOS-11 volumes and output volumes default to Files-11 format. FLX assumes these default volume formats if you do not specify a format qualifier in the file transfer command string.

You can change the default by entering /DO or /RS on a command line by itself. To specify that the default transfer is from DOS-11 to Files-11, type:

```
FLX>/DO
```

To specify that the default transfer is from Files-11 to DOS-11, type:

```
FLX>/RS
```

If /RT is specified on one side of a command string, the default qualifier for the other side is /RS. For example:

```
FLX>DM0:/RT=DD0:SYS1.MAC
```

The input is defaulted to /RS. In the next example, the output is defaulted to /RS:

```
FLX>DM0:=DM0:SYS1.MAC/RT
```

You cannot transfer files directly between RT-11 and DOS-11 volumes using FLX.

2.3.2 Transfer Mode Qualifiers

Transfer mode qualifiers are used to specify the format that an output file should have after it is copied or converted. FLX has three transfer mode qualifiers, one for each type of file format: formatted ASCII, formatted binary, and file image format. Format conversions can be in either direction between DOS-11 files and Files-11 files or between RT-11 files and Files-11 files. Table 2-2 describes the transfer mode qualifiers.

Table 2-2
FLX Transfer Mode Qualifiers

Qualifier[1]	Function
/FA:n	Formatted ASCII Formatted ASCII files consist of ASCII data records terminated by carriage return/line feed (CR-LF), form feed (FF), or vertical tab (VT) characters. In transfers from DOS-11 or RT-11 files to Files-11 files, CR-LF pairs are removed from the end of records. In transfers from Files-11 files to DOS-11 or RT-11 files, CR-LF pairs are added to the end of each record that does not already end with LF or FF. All null, rubout, and vertical tab characters are removed from input records in any of these transfers. If you specify /FA:n with Files-11 output, fixed-length records of size n are generated. Output records are padded with null characters as necessary. If you do not specify /FA:n with Files-11 output, FLX generates variable-length records. The output record size equals the input record size. ASCII data is transferred as 7-bit codes. Bit 8 (the parity bit) of each byte is masked before transfer.

(continued on next page)

1. Note that n, which specifies record size, is useful only for Files-11 volumes. If you enter it when specifying other volumes, it will be ignored. All n values are interpreted in octal unless followed by a period.

Table 2-2 (Cont.)
FLX Transfer Mode Qualifiers

Qualifier[1]	Function
/FB:n	Formatted Binary If you specify /FB with DOS-11 or RT-11 output files, formatted binary headers and checksums are added to the records. Specifying /FB:n with Files-11 output produces fixed-length records of size n, up to 512 decimal bytes long. FLX pads records with null characters to reach the specified length. If you do not specify n for Files-11 output, FLX generates variable-length records. The output record size equals the input record size.
/IM:n	Image Mode Specifying an image mode transfer always produces fixed-length records. You can use the value n to indicate the desired number of bytes in the record (up to 512) for Files-11 output. If you do not specify n, FLX assumes a record length of 512 bytes.

1.Note that n, which specifies record size, is useful only for Files-11 volumes. If you enter it when specifying other volumes, it will be ignored. All n values are interpreted in octal unless followed by a period.

FLX assumes the following default transfer modes for these file types:

Qualifier	File Type
/IM:n (Image Mode)	TSK, OLB, MLB, SYS, SML, ULB, EXE
/FB (Formatted Binary)	OBJ, STB, BIN, LDA
/FA (Formatted ASCII)	All others

2.3.3 Control Qualifiers

FLX provides control qualifiers to control file processing. Table 2-3 describes these qualifiers.

Table 2-3
FLX Control Qualifiers

Qualifier	Function
/BL:n	Indicates the number of contiguous blocks (n) to be allocated to the output file. If you do not specify /BL, the input file size is used as the output file size. /BL:n is used with RT-11 output to circumvent the normal RT-11 file allocation scheme, which allocates the largest available space on the volume for a new file. Using /BL:n with the /RT switch for the output file causes n blocks to be allocated for the output file instead of the largest available space. When FLX has finished transferring the file to the RT-11 volume and the file is closed, the output file will have the same number of blocks as the input file, less than or equal to n. If the input file size is larger than n, an error will occur. The /BL:n qualifier is normally used with the /CO qualifier, as described below. Because all RT-11 files are contiguous, the /CO qualifier need not accompany the /RT:n qualifier for RT-11 output.
/CO	Indicates that the output file is to be contiguous. The /CO qualifier is used only with disk output. When the input file is in DOS-11 format, use /BL:n with /CO (see the description of /BL:n above). When the input is a Files-11 volume or an RT-11 disk, FLX assumes /CO in transferring file types TSK, SYS, and OLB to Files-11 volumes.
/DE	With /RT, deletes files from a disk with RT-11 formatted volumes. When you specify /DE, the FLX command string needs no output specification.
/DI or /LI	Causes a directory listing to be listed on a specified output file. Use /RT with /DI or /LI to generate a directory listing of RT-11 volumes. If you do not specify an output device, the directory will be sent to SYS$OUTPUT. If you do not specify file name and file type on the input file specifications, *.* is assumed. You cannot list Files-11 volume directories using FLX.

(Continued on next page)

Table 2-3 (Cont.)
FLX Control Qualifiers

Qualifier	Function
/DNS:n	Specifies the magnetic tape density in bits per inch (bpi); n is either 800 or 1600. If n is any other number or is not specified at all, an error will occur. If you do not specify /DNS:n, the magnetic tape density will default to 800 bpi. If you specify /DNS with any device but magnetic tape, FLX will ignore the qualifier.
/FC	With FORTRAN files on Files-11-formatted output files, indicates that FORTRAN carriage control conventions should be used, that is, that FORTRAN should interpret certain characters as carriage control characters. (See the VAX-11 FORTRAN Language Reference Manual for more information on FORTRAN carriage control conventions, or see the VAX-11 Record Management Services Reference Manual for a discussion of the file access block and record attributes, which include setting carriage control.)
/ID	Requests that the current version number of FLX be printed. You can specify /ID as part of an output or input specification, or type it in response to the FLX prompt (FLX>).
/LI	Same as /DI, explained above.
/NU:n	With the /ZE and /RT qualifiers, specifies the number of directory blocks (n) to be allocated when FLX initializes an RT-11 disk. If you do not specify /NU:n, four directory blocks will be allocated. The maximum number of blocks that can be allocated is 31.
/RW and /-RW	/RW rewinds the magnetic tape before FLX begins the file transfer. /-RW causes FLX to begin the transfer without first rewinding the magnetic tape. The default is /RW. If you specify /RW or /-RW with any device other than magnetic tape, or with the qualifiers /DI, /LI, or /ZE, FLX will ignore the rewind qualifier.
/SP	Indicates that the converted file is to be spooled. /SP is used only with Files-11 output files.
/UI	Indicates that the output file is to have the same directory as the input file. Do not use /UI when you are specifying an explicit output UIC. /UI is valid only with output files in DOS-11 or Files-11 format.
/ZE	Initializes DOS-11 or RT-11 volumes. To initialize RT-11 volumes, you must also specify /RT and /NU:n. Initializing erases any files already on the device.

2.4 TRANSFERRING FILES WITH FLX

To transfer files from one volume to another, enter a command string of the form

```
device-spec[/qualifier]=file-spec[/qualifier]...file-spec[/qualifier]
```

The FLX transfer specifies the output device on the left of the equal sign and the files to be transferred on the right of the equal sign.

In constructing the transfer command string, keep in mind the restrictions upon the various FLX qualifiers listed in Section 2.3, as well as the restrictions upon the format of device and file specifications.

2.5 DOS-11 VOLUME DIRECTORY MANIPULATION

You can display DOS-11 directory listings and initialize DOS-11 volumes using FLX qualifiers as described in the following sections.

Remember that DOS-11 volumes must be mounted foreign before you can manipulate them using FLX.

2.5.1 Displaying DOS-11 Directory Listings

The /DI qualifier described in Table 2-3 sends the directory of the DOS-11 volume specified in the input specification to the Files-11 file specified in the output specification. If you do not enter an output specification, FLX sends the directory to SYS$OUTPUT. For example:

```
FLX>MT0:/DO/DI
```

This command lists on your terminal all files from the DOS-11 volume on the magnetic tape drive MT0.

Figure 2-1 shows a sample directory listing of a DOS-11 volume, followed by notes keyed to the figure.

```
DIRECTORY (1)      MT0:[360,273] (2)
29-JUN-79 (3)

                                                    (6)
FLXCHA.DOC (4)              35. (5)    29-JUN-79
PRACTW.DOC                  2.         29-JUN-79
WATSP.DOC                   3.         29-JUN-79

TOTAL OF 40.  BLOCKS IN 3.  FILES (7)
```

Figure 2-1 DOS-11 Directory Listing

Notes to Figure 2-1:

(1) The listing identifier.

(2) The device name, unit number, and UFD.

(3) The date the directory was listed.

(4) The file name and file type.

❺ The number of blocks in the file.

❻ The file creation date.

❼ The total number of blocks allocated to all files on the volume.

2.5.2 Initializing DOS-11 Volumes

You can initialize DOS-11 volumes using the /ZE qualifier. This qualifier requires only the device specification for the volume you are initializing. For example:

```
FLX>MT0:/DO/ZE
```

This command initializes the magnetic tape on MT0 in DOS-11 format.

2.6 RT-11 VOLUME DIRECTORY MANIPULATION

You can display RT-11 directory listings, delete RT-11 files, and initialize RT-11 volumes using FLX qualifiers as described in the following sections.

Remember that VAX/VMS RT-11 volumes must be mounted foreign before you can manipulate them using FLX.

2.6.1 Displaying RT-11 Directory Listings

The /DI qualifier, when combined with the /RT qualifier, sends the directory of the RT-11 volume specified in the input specification to the Files-11 file specified in the output specification. If you do not enter an output specification, FLX will send the directory to SYS$OUTPUT. For example:

```
FLX>DM0:*.MAC/DI/RT
```

This command lists on your terminal all files with the file type of MAC from the RT-11 volume on DM0.

Figure 2-2 shows a sample directory listing of an RT-11 volume, followed by notes keyed to the figure.

```
DIRECTORY ❶      DM0 ❷
22-JUN-79 ❸
                                          ❻
SIPBOO.MAC ❹     49. ❺    22-JUN-79
< UNUSED >        6.
SIP    .MAC      10.      22-JUN-79
SIPCD  .MAC       7.      22-JUN-79
< UNUSED >       21.
SIPQIO.MAC        7.      22-JUN-79
< UNUSED >    26998.

 27025.  FREE BLOCKS ❼
                                          ❽
TOTAL OF 73.  BLOCKS IN 4.  FILES
```

Figure 2-2 RT-11 Directory Listing

Notes to Figure 2-2:

❶ The listing identifier.

❷ The device name and unit number.

❸ The date the directory was listed.

❹ The file name and file type; < UNUSED > indicates free space.

❺ The number of blocks in the file or free space.

❻ The file creation date; blank for free space.

❼ The total number of free blocks on the volume.

❽ The total number of blocks allocated to all files on the volume.

2.6.2 Initializing RT-11 Volumes

You can initialize RT-11 volumes using the /ZE qualifier with the /RT qualifier. The /ZE qualifier requires only the device specification for the volume you are initializing. For example:

```
FLX>DM1:/ZE/RT
```

This command initializes the RT-11 formatted volume on DM1.

When you initialize RT-11 volumes, /ZE takes an optional argument in the form:

```
/ZE:n
```

The value n specifies the number of extra words per directory entry, in addition to the 7-word default length. This capacity for increasing the length of directory entries is useful for some RT-11 applications. Note that when you increase the number of words per directory entry by specifying /ZE:n, you are reducing the number of directory entries.

Using the /NU:n qualifier with /ZE and /RT specifies the number of directory segments, n, to be allocated to the RT-11 volume. Four directory segments (consisting of two disk blocks each) are allocated by default. The maximum number of segments that can be allocated is 31(10). For example:

```
FLX>DM0:/ZE:2/NU:6/RT
```

This command

- Initializes the disk on DM0
- Allocates two extra words per directory entry
- Allocates six directory segments

2.6.3 Deleting RT-11 Files

You can delete files from RT-11 disks using the /DE qualifier with the /RT qualifier. The command string on which you specify /DE/RT requires the device and file specifications for the file you are deleting. For example:

```
FLX>DM1:SYS1.MAC/DE/RT
```

This command deletes the file SYS1.MAC from the RT-11 volume on DM1.

2.7 FLX MESSAGES

Errors encountered by FLX during processing are reported on the initiating terminal. The FLX messages, their explanations, and suggested user actions are described below.

FLX -- BAD LIST FILE SPEC

Explanation: One of the following was specified for a /DI operation (directory listing):

- More than one output file
- Wild card characters in the output file

User Action: Reenter the command line correctly.

FLX -- CAN'T OPEN @ FILE

Explanation: The specified indirect command file could not be opened for one of the following reasons:

- The file is protected against access.
- A problem exists on the physical device (for example, the disk is not spinning).
- The volume is not mounted or is allocated to another user.
- The specified file directory does not exist.
- The named file does not exist in the specified directory.
- The volume is not online.

User Action: Correct the condition and reenter the command line.

FLX -- CO FILES TO OUTPUT DEVICE NOT ALLOWED

Explanation: An output device (for example, a magnetic tape) was entered with the /CO qualifier for which the /CO qualifier is not valid.

User Action: Reenter the command line without specifying /CO.

FLX -- COMMAND SYNTAX ERROR

Explanation: The command was entered in a format that does not conform to syntax rules.

User Action: Reenter the command line with the correct syntax.

FLX -- CONFLICTING TRANSFER MODES SPECIFIED

Explanation: Conflicting transfer mode qualifiers were entered. For example:

```
SY:=DM:FOO.OBJ/IM/FB
```

User Action: Reenter the command line with only one transfer mode qualifier specified.

FLX -- DOS-11 OR RT-11 DEVICE NOT VALID FORMAT

Explanation: The device specified with /DO has an incorrect DOS-11 file structure, or the device specified with /RT has an incorrect RT-11 file structure.

User Action: Correctly identify the file structure on each volume, and reenter the command line.

FLX -- ERROR DURING DIRECTORY I/O

Explanation: One of the following conditions may exist:

1. The volume is not write-enabled.
2. The volume format qualifiers (/DO, /RT or /RS) were incorrectly specified.
3. The volume is not of the proper format.
4. A hardware error occurred during a directory I/O operation (for example, a bad tape).

User Action: The following responses correspond (by number) to the conditions listed above.

1. Write-enable the volume.
2. Respecify the volume format qualifiers (/DO, /RT, or /RS) correctly.
3. No recovery is possible with the volume currently mounted. Mount a volume that is in the proper format, and retry the operation.
4. Retry the operation.

FLX -- FILE NOT FOUND

Explanation: The named file does not appear as specified in the requested directory.

User Action: Retry the operation with the file name and directory correctly specified.

FLX -- @ FILE NESTING EXCEEDED

Explanation: More than one level of indirect command file was specified.

User Action: Retry the operation with only one level of indirect command file specified.

FLX -- @ FILE SYNTAX ERROR

Explanation: A syntax error occurred in the indirect command file specification.

User Action: Edit the indirect command file. Rerun FLX using the corrected indirect command file.

FLX -- FMTD ASCII RECORD FORMAT BAD

or

FLX -- FMTD BINARY RECORD FORMAT BAD

Explanation: Either the file is corrupted, or the file is not of the specified type.

User Action: If the file is corrupted, no recovery is possible. If the file type is incorrect, retry the operation specifying the correct transfer mode switch.

FLX -- INCORRECT # IN/OUT SPECS

Explanation: More than one input or output specification in a command was entered where only one is allowed.

User Action: Reenter the command line with the proper syntax.

FLX -- INVALID DEVICE

Explanation: A device was specified that cannot be used for the purpose specified; for example, a line printer was specified as an input device.

User Action: Reenter the command line with a legal device specified.

FLX -- INVALID DOS OR RT-11 FILE SPEC

or

FLX -- INVALID RSX FILE SPEC

Explanation: The file specification does not conform to proper syntax, or the specified operation could not be performed on the specified device.

User Action: Reenter the file specification with the proper syntax.

FLX -- INVALID SYNTAX

Explanation: A qualifier was entered that is not a valid FLX qualifier or does not conform to proper syntax.

User Action: Reenter the command line with a correct qualifier specification.

FLX -- I/O ERROR

Explanation: One of the following conditions may exist:

- The specified device is offline.
- A hardware error occurred (for example, a bad tape).

User Action: Ensure that the device is online. Reenter the command line. If a hardware error occurred, recovery may not be possible.

FLX -- I/O ERROR INITIALIZING DIRECTORY

Explanation: One of the following conditions may exist:

- The specified directory is not online.
- The specified volume is not mounted.
- A hardware error occurred (for example, a bad tape).

User Action: Ensure that the device is online and is operable. Reenter the command line with the required qualifier specified. If a hardware error occurred, recovery may not be possible.

FLX -- I/O ERROR ON COMMAND INPUT

Explanation: An unexpected error in command input was encountered from either an indirect command file or your terminal; FLX exits.

User Action: Restart FLX.

FLX -- I/O ERROR ON FLX TEMPORARY FILE

Explanation: FLX encountered an error condition with its temporary file. FLX creates a temporary file on your default disk for operations involving DOS-11 magnetic tape. This error occurs when one of the following conditions exists:

- Your default disk is not online and mounted.
- Your default disk is write-locked.
- A protection violation occurred.
- A hardware error was encountered.

User Action: Correct the condition and reenter the command line.

FLX -- I/O ERROR ON LIST FILE

Explanation: An error occurred on the output device during a /DI or /LI sequence. There is a hardware problem with the output device (for example, device powered down).

User Action: Correct the condition and reenter the command line.

FLX -- OUTPUT DEVICE FULL

Explanation: The DOS-11 or RT-11 output volume does not contain enough space for the output file.

User Action: Delete all unnecessary files and reenter the command line.

FLX -- OUTPUT FILE SPEC NOT ALLOWED

Explanation: An output file specification was entered for a command that does not allow one.

User Action: Reenter the command without an output file specification.

FLX -- RECORD TOO LARGE

Explanation: FLX detected an input record in a Files-11 transfer that is larger than the specified or implied record size for the file, that is, the file is corrupted.

User Action: The file in question is unusable.

FLX -- WARNING -- SPECIFIED RECORD SIZE BAD, 512. USED

Explanation: The record size n specified with /FA:n, /FB:n, or /IM:n is not acceptable. A record size of 512(10) bytes is assumed.

User Action: This is a warning message. No action is required.

FLX --UNABLE TO ALLOCATE FILE

Explanation: No space is available on the DOS-11 or Files-11 volume for the specified file.

User Action: Delete all unnecessary files and reenter the command line.

FLX -- UNABLE TO OPEN FILE

Explanation: A specified Files-11 input or output file could not be opened. Possible reasons are:

- The input file does not exist.
- The volume is not mounted.
- A protection violation occurred.

User Action: Correct the condition and reenter the command line.

FLX -- UNABLE TO OPEN LIST FILE

Explanation: The directory listing file cannot be opened under the specified file name and directory, or the specified device may not be a valid Files-11 volume.

User Action: Reenter the command line specifying the correct file name and directory.

FLX -- UNDIAGNOSABLE REQUEST

Explanation: FLX does not recognize the command line syntax.

User Action: Reenter the command line with the proper syntax.

FLX -- /CO FILES FROM INPUT DEVICE NOT ALLOWED UNLESS /BL: SPEC

Explanation: When transferring files from magnetic tape, /CO can only be specified when /BL is also specified.

User Action: Reenter the command line, specifying /BL.

FLX -- * IN OUTPUT UIC NOT ALLOWED

Explanation: A wild card character was detected in the user identification code for the output volume.

User Action: Reenter the command line without wild card characters in the output specifications.

FLX -- * IN VERSION NUMBER NOT ALLOWED

Explanation: A wild card character was detected in the version number field of a file specification.

User Action: Reenter the command line with all version numbers explicitly specified.

CHAPTER 3

SLP AND SUMSLP EDITING UTILITIES

Two batch-oriented text editors run on VAX-11 processors: SLP and SUMSLP. To use either of these editing utilities, you generate a list of the changes which you want to apply to your source file. The utilities effect these changes and produce an edited output file. This chapter describes how to use these editors.

3.1 SLP

SLP is a batch-oriented editing program used for source file maintenance. The term "SLP" originally meant "source language input program." SLP allows you to update (delete, replace, add) lines in an existing file. Furthermore, SLP gives you a record (audit trail) of editing changes. The SLP command file provides a reliable way to duplicate the changes made to a file, at a later time or on another computer system.

Input to SLP consists of (1) an input source file that you want updated, and (2) a command file containing text lines and edit command lines that specify the update operations to be performed. SLP locates lines to be changed by means of "locators" (sequence numbers or character strings).

SLP output is an updated copy of the input source file. SLP provides an audit trail that helps you keep track of the update status of each line in the file. The audit trail is included permanently in the output file. When a given file is updated with successive versions of SLP command file, you can use different audit trails to differentiate among the changes made at different times.

SLP output qualifiers modify the appearance of the output file. They let you truncate lines, create or suppress an audit trail, eliminate an existing audit trail, create checksums, and specify the length and beginning position of the audit trail.

3.1.1 Invoking SLP

You can run SLP either indirectly or interactively. In either case, you invoke SLP with the command line:

```
EDIT/SLP [/qualifiers(s)] infile-spec
```

qualifier(s)

Actions to be performed by SLP that control the generation and format of the listing and output files (see Section 3.1.4).

infile-spec

The input source file specification (see Section 3.1.2.1).

When you run SLP indirectly from a SLP command file, this command line, preceded by a dollar sign ($), is the first line of the file, as described in Section 3.1.2.2.

When you run SLP interactively, this command line is typed in response to the DCL prompt ($), as described in Section 3.1.3.

3.1.2 Running SLP Indirectly

SLP requires two types of input files: an input source file and a SLP command file. These files are described in Sections 3.1.2.1 and 3.1.2.2.

The output file, described in Section 3.1.2.3, is the permanently updated copy of the input file. It shows the changes SLP makes to the input file.

You can also generate a listing file, described in Section 3.1.2.3.

Figure 3-1 shows the relationships among the SLP output and input files. The contents of the various files in this figure are described in the following sections.

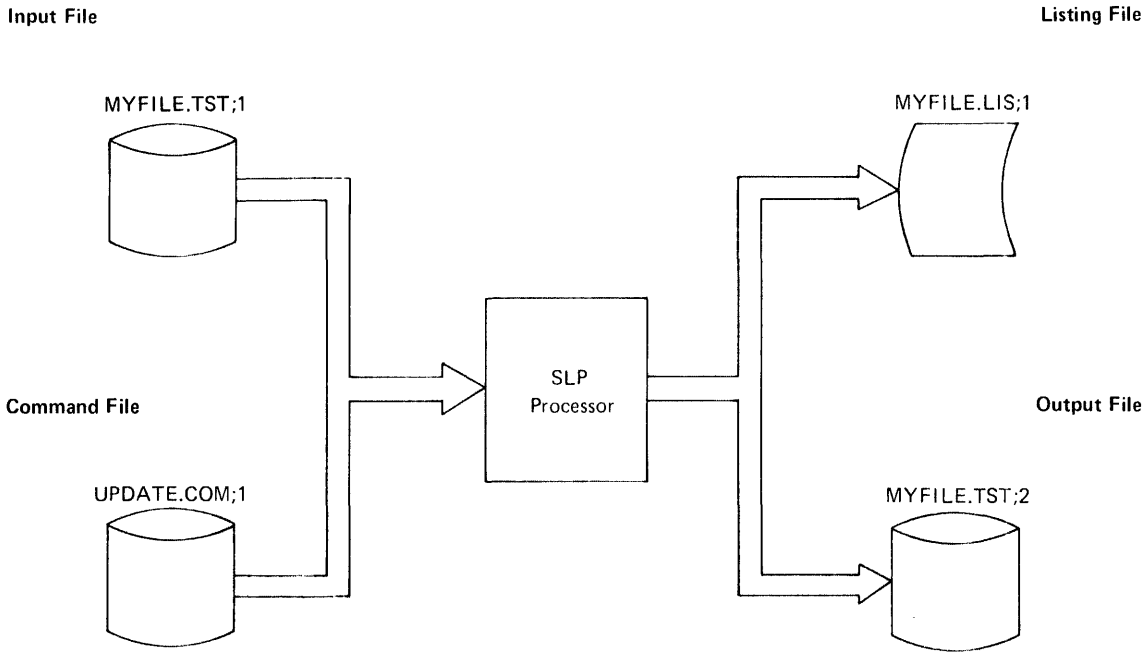

Figure 3-1 Files Used During SLP Processing

3.1.2.1 **The Input Source File** - The input source file is the file to be updated by SLP. It can contain any number of lines.

To use SLP effectively, you should obtain a sequence-numbered listing of the input file from which you can determine what editing commands you will issue. Section 3.1.5.3 describes how to generate such a listing using the /LIST qualifier. However, the input source file actually updated by SLP can have any kind of line numbers.

3.1.2.2 **The SLP Command File** - The SLP command file is a VAX/VMS file that contains SLP editing commands. It consists of four elements:

1. An initialization line that invokes SLP and specifies what file to process:

   ```
   $ EDIT/SLP [/qualifiers] infile-spec
   ```

   This command line is described in Section 3.1.1.

2. SLP editing command lines that define changes to the input file (see Section 3.1.6).

3. Input lines, that is, lines of text that are to be inserted into the output file, either as new lines or to replace old lines.

4. The SLP terminator, a single slash (/) in column 1, that causes SLP to begin its processing (updating) of the file.

An interactive text editor is usually used to create SLP command files. Once you have created the file, you can submit it for processing by using the DCL commands Execute Procedure (@) or SUBMIT.

The example below shows a SLP command file named UPDATE.COM. The numbers to the right of the example correspond to the elements listed above.

```
$ EDIT/SLP MYFILE.TST                              (1)
-3                                                 (2)
INSERT THIS LINE AFTER LINE 3                      (3)
-4,4                                               (2)
DELETE LINE 4 AND REPLACE IT WITH THIS LINE        (3)
/                                                  (4)
```

You can execute this file by using the Execute Procedure (@) command, as follows:

```
$ @UPDATE
```

Because the file type is the default file type COM, it can be omitted on the DCL command line. See the VAX/VMS Command Language User's Guide for information on using command procedures and running batch jobs.

When SLP finishes its processing, the DCL prompt is issued:

```
$
```

You can request that SLP calculate a checksum value for SLP editing commands and then use this value to determine whether you have made the correct changes to your source file. See Section 3.1.4.2 for a description of calculating checksums.

3.1.2.3 **The Output File** - The SLP output file is the updated input file. All of the updates specified by the SLP editing commands are inserted in this file. An audit trail is applied, by default unless suppressed, to new or changed lines (see Section 3.1.4). You can also specify the text and length of an audit trail (see Section 3.1.6.6).

An output file is generated by default. You can suppress the output file, however, by using the /NOOUTPUT qualifier, described in Section 3.1.4.2.

3.1.2.4 **The Listing File** - You can generate a listing file by using the /LIST qualifier, described in Section 3.1.4.3. The listing file shows the changes made to the source file. Each line in the listing file shows the updates made to the source file. Each line in the listing is numbered in sequence. Updates are indicated by means of an audit trail (unless you suppress audit-trail generation). Section 3.1.5 contains an example of a listing file.

A sequence-numbered listing of the input file can help you determine what editing commands to use. Generating this listing is described in Section 3.1.4.3.

3.1.3 Running SLP Interactively

To use SLP interactively, type the following command line in response to the DCL prompt:

```
$ EDIT/SLP [qualifier(s)] infile-spec
```

If you do not enter the input source file specification, you will be prompted for it with the prompt:

```
File:
```

After specifying the input source file specification, enter SLP editing commands, one per line. Then enter the SLP terminator in the first column of the next line. The utility will respond to the terminator with the prompt:

```
SLP>
```

To end the interactive SLP editing session, type <CTRL/Z> in response to the SLP> prompt.

3.1.4 How SLP Processes Files

This section uses an example to show how SLP processes files. It uses the following source input file, named MYFILE.TST;1.

```
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
TEN
```

This file is to be updated under the control of the following SLP command file, named UPDATE.COM. The editing commands used in the command file are described in Section 3.1.6.

```
$EDIT/SLP/AUDIT_TRAIL:50/LIST  MYFILE.TST
-3
INSERT THIS LINE AFTER LINE 3
-4,4
DELETE LINE 4 AND REPLACE IT WITH THIS LINE
/
```

Below is the listing file (MYFILE.LST) that results from issuing the command @UPDATE.

```
MYFILE.TST/AU:50.:10.,MYFILE=MYFILE.TST

    1.  ONE
    2.  TWO
    3.  THREE
    4.  INSERT THIS LINE AFTER LINE THREE                    ;**NEW**
    5.  DELETE LINE 4 AND REPLACE IT WITH THIS LINE          ;**NEW**
    6.  FIVE                                                 ;**-1
    7.  SIX
    8.  SEVEN
    9.  EIGHT
   10.  NINE
   11.  TEN
```

The audit trail, using the default audit trail texts described in Section 3.1.5, shows the new lines (;**NEW**) and indicates where lines have been removed (;**-1). In this case, a new line has been added after line 3, and line 4 has been replaced, causing all subsequent lines to be renumbered. The /AUDIT_TRAIL qualifier in the initialization line indicates that the audit trail is to begin at the next tab stop after column 50.

To process the files, SLP writes each line from the input source file into the output file until it reaches a line to be modified, as requested in the command. When SLP reaches a line to be modified, it makes the indicated modification, notes the change in the audit trail, and then continues writing lines to the output file, in sequence, until it encounters another command or reaches the end of the source file.

The output file, MYFILE.TST;2, is as follows:

```
ONE
TWO
THREE
INSERT THIS LINE AFTER LINE 3                    ;**NEW**
DELETE LINE 4 AND REPLACE IT WITH THIS LINE      ;**NEW**
FIVE                                             ;**-1
SIX
SEVEN
EIGHT
NINE
TEN
```

3.1.5 SLP Qualifiers

SLP qualifiers control the generation and format of the listing file and the output file. You can use them to control the audit trail and output options associated with these files. Table 3-1 describes the SLP qualifiers and their functions. The following sections illustrate the use of SLP qualifiers.

Table 3-1
SLP Qualifiers

Format	Function
/AUDIT_TRAIL [:(POSITION:pos,SIZE:len)] and /NOAUDIT_TRAIL	These qualifiers let you suppress audit-trail generation, or specify the beginning position and length of the audit trail. The default is to generate an audit trail 8 characters long, starting in column 80 -- that is, /AUDIT_TRAIL:(POSITION:80,SIZE:8). The maximum allowed value for the length parameter is 16. The audit trail starts at the first tab stop after the position given (or defaulted) for the /AUDIT_TRAIL qualifier. Tab stops are set every 8 columns.
/CHECKSUM [:n] and /NOCHECKSUM	The /CHECKSUM qualifier requests a checksum calculation for SLP edit commands. If you do not specify a checksum value, SLP reports the calculated checksum at your terminal. If you do specify a value, SLP will generate a diagnostic message if the specified values does not match the checksum calculation. The default is /NOCHECKSUM.
/LIST [:list-file]	The /LIST qualifier causes SLP to produce a sequentially numbered version of the input file with the same file name. The default file type is LST. You can request a different specification for the listing file by using the /LIST:list-file qualifier.

(continued on next page)

Table 3-1 (Cont.)
SLP Qualifiers

Format	Function
/OUTPUT [:file-spec] and /NOOUTPUT	By default, SLP generates an output file with the same file name and file type as the correction input file. Its version number is higher by 1 than the highest version number existing for the input file name and type. You can request a different specification for the output file by using the /OUTPUT:file-spec qualifier. To suppress the output file, specify /NOOUTPUT.
/REPORT and /NOREPORT	The qualifier /REPORT causes SLP to report any line truncation by audit trails. If line truncation occurs, SLP prints a diagnostic message. If you specify creation of a listing file, a question mark (?) replaces the period (.) in the line number of the truncation line. The default is /NOREPORT.
/TAB_FILL and /NOTAB_FILL	The qualifier /TAB_FILL causes SLP to insert tabs at the end of each text line containing an audit trail. The default is to fill such lines with spaces (that is, /NOTAB_FILL). Using /TAB_FILL saves disk space, because fewer tabs than spaces are required to fill the lines in both the output and the listing file.
/TRUNCATE [:position] and /NOTRUNCATE	The /TRUNCATE qualifier lets you truncate input lines to the given column position. If you specify /TRUNCATE but omit position, SLP uses the position given (or defaulted) for the /AUDIT_TRAIL qualifier; position is rounded to the next tab stop before use. Set position at or before the start of the old audit trail that you want to delete. Any trailing spaces or tabs after position are also deleted.

3.1.5.1 **Using the /AUDIT_TRAIL Qualifier** - You may want to change the position of the audit trail if your output device has fewer than 80 columns or if your source lines are all brief. The following example shows the use of the /AUDIT_TRAIL qualifier to specify the position

and length of the audit trail. By default, audit trail texts are ;**NEW** for new lines and ;**-n for deleted lines. (See Section 3.1.6.6 for a description of changing the audit trail text.) The input source file for this example is named MYFILE.TST and is made up of the following lines:

```
ONE
TWO
THREE
FOUR
FIVE
```

The SLP command file is as follows:

```
$EDIT/SLP/AUDIT_TRAIL:(POSITION:30,SIZE:10)/LISTING  MYFILE.TST
-2,,+1,/!CHANGE001/
NEW LINE 2
NEW LINE 3
/
```

The following listing file results from SLP processing.

```
MYFILE.TST/AU:30.:10.,MYFILE=MYFILE.TST


     1.  ONE
     2.  NEW LINE 2                    !CHANGE001
     3.  NEW LINE 3                    !CHANGE001
     4.  FOUR                          !**-2
     5.  FIVE
```

The values that you specified for position and length are stated in the header of the listing file.

3.1.5.2 **Using the /CHECKSUM Qualifier** - To obtain a checksum value, append the qualifier /CHECKSUM to the SLP initialization line. SLP processes the file, prints the checksum value in a message on your terminal, and exits.

If you want to check the accuracy of SLP editing commands, specify a checksum value using the form /CHECKSUM:n, where n is the checksum value previously calculated by SLP. If there is a mistake in the SLP command file (for example, if the edit command is -4,4 and you type -4,5), the checksum value will not match the value you specified with the /CHECKSUM qualifier. If the two values differ, SLP prints a diagnostic error message on your terminal, as described in Section 3.3.1.

SLP calculates a checksum value for all SLP edit commands except:

- The SLP initialization line
- Comments within the edit command line
- Spaces and/or tabs between characters included in the checksum calculation and those characters excluded from the calculation
- The second comma on an edit command line and anything following it (that is, audit trails and comments)
- The comment delimiter for an input line and any characters following it (the comment delimiter is defined as the first character in the current audit trail).

3.1.5.3 **Using the /NOOUTPUT and /LIST Qualifiers** - SLP processes input by sequence number. However, sequence numbers appear only in the listing file; they are not written to the output file.

To use SLP effectively, obtain an up-to-date numbered listing for use when you create the SLP command file. Numbered listings generated by other programs (such as SOS and the MACRO assembler) will not necessarily be useful in preparing an SLP command file. Generate a SLP numbered listing by submitting an editing command in the following form:

```
$EDIT/SLP/NOOUTPUT/LIST[:list-file]  file-spec
```

Here list-file is the name you optionally assign to the listing file that SLP produces, and input-file is the specification of the file whose lines are to be numbered. The slash (/) tells SLP to begin processing the files. SLP generates a numbered listing file, but does not produce an output file.

3.1.6 Specifying SLP Editing Commands

SLP editing commands let you update source files by adding, deleting, and replacing lines in a file. These commands contain certain characters that SLP interprets as operators. This section first describes these operators and the general form for specifying SLP editing commands. Then, it describes the editing commands used for specific editing functions.

3.1.6.1 **SLP Operators** - When SLP encounters any of the characters listed in Table 3-2 as the first character in an input line, it interprets the character as an operator.

Table 3-2
SLP Operators

Operator	Function
- (minus sign)	First character of an SLP edit command
\ (backslash)	Suppress audit trail generation
% (percent sign)	Reenable audit trail generation
@ (at sign)	Invoke a further command file for SLP processing
/ (slash)	Terminate the editing session
< (less than character)	Escape character

The percent sign (%) operator is used to reenable audit trail generaton when generation has been suppressed by either the backslash (\) operator or the /NOAUDIT_TRAIL qualifier, described in Section 3.1.5.

The at sign (@) operator tells SLP to read further input from a another command file. This second command file can contain only SLP edit commands and new text lines.

The less-than character (<) operator is the escape character that lets you enter characters in the command file (in column 1) that SLP otherwise would interpret as operators. For example, </ hides the slash character from SLP, thereby enabling you to enter the slash into the output file without terminating the SLP editing session. You can use the less-than character as an escape character for all SLP operators listed in Table 3-2 (including itself).

3.1.6.2 **General Form of an Editing Command** - The general form of a SLP editing command is:

```
-locator1[,locator2][,/audittrail/][;comment]
inputline
     .
     .
     .
```

-

A minus-sign operator indicates that this is an SLP editing command line.

locator1

A line locator that causes SLP to move the current line pointer to a specified line. If only locator1 is specified, the current line pointer is moved to that line and SLP reads the next line in the editing command file. This field can be specified using any of the locator forms described below.

locator2

A line locator that defines a range of lines (that is, the range beginning with locator1 and ending with locator2) to be deleted or replaced. This field can be specified using any of the locator forms described below.

/audittrail/

A character string used to keep track of the update status of each line in the file. This audit trail is used to mark new or replaced lines in the file until the audit trail is either changed or suppressed. This argument must be delimited by slashes (/). If there are not two locator fields in the editing command, the audit trail specification must be preceded by two commas.

Audit trails generated by SLP use the first character of the specified string as a delimiter. Usually, the first character of the audit trail is set to match the comment delimiter of the source file being edited. Default audit trails are ;**NEW** for new lines and ;**-n for lines that indicate where text has been inserted.

inputline
A line of new text to be inserted into the file immediately following the current line. You can enter any number of input lines.

;comment
An optional comment. SLP ignores any text after a semicolon.

All fields in the command line are position-dependent; commas must be included as specified above.

The locator fields can take one of the following forms:

$$\left\{\begin{array}{c}/string[...string]/\\ number\\ .\end{array}\right\} \quad [+n]$$

Parameters:

string
A string of ASCII characters. SLP locates the next line in which string exists and moves the current line pointer to that line. If the locator is specified in the form /string...string/ (that is, two different strings of characters separated by three periods), SLP locates the line in which the first character string is followed by the second character string, regardless of what characters may be in between them.

number

A sequence number in the range of 1 through 9999 to which the current line pointer is to be moved.

n

A decimal value used as an offset from the line specified by the locator. Note that n is always preceded by a plus sign (+). You cannot back up from the locator.

.

A period represents the current line.

All forms of the line locator can be specified interchangeably in a command line.

SLP can only edit files sequentially. Once the current line pointer moves past a given line in the file, it cannot be returned to that line.

3.1.6.3 **Adding Lines to a File** - The SLP editing command for adding lines to a file contains only one locator field. Its form is:

-locator[,,/audittrail/][;comment]

The locator has one of the forms defined in Section 3.1.5.2

If a numeric locator is specified, SLP inserts new line(s) after the line specified by sequence number. Any lines you enter are inserted as lines in the file.

If a string locator is specified, SLP locates the next occurrence of the string in the file and moves the current line pointer to the line containing the string. Any input lines following the command line are then added to the file.

If you specify an offset (+n) SLP moves the current line pointer n lines beyond the line specified in the locator field and then adds any new input lines to the file.

Because there is only one locator field, the audit trail specification must be preceded by two commas.

The example below shows how to add lines to a file. The input source file consists of the following lines:

```
ABC
DEF
GHI
KLM
123456789
456
789
CBA
XYX
987
```

The SLP command file consists of the following commands and text lines:

```
$EDIT/SLP/LISTING/AUDIT_TRAIL:(POSITION:32) MYFILE.TST
-/123/
INSERT THIS LINE AFTER LINE 5
/
```

SLP processing generates the following listing file:

```
MYFILE.TST;MYFILE=MYFILE.TST

     1.  ABC
     2.  DEF
     3.  GHI
     4.  KLM
     5.  123456789
     6.  INSERT THIS LINE AFTER LINE 5        !**NEW**
     7.  456
     8.  789
     9.  CBA
    10.  XYX
    11.  987
```

SLP has applied sequence numbers to the lines and added an audit trail to the line following line 5, where SLP found the first occurrence of the string 123.

The next example uses the same correction input file and the following new SLP command file:

```
$EDIT/SLP/LISTING/AUDIT_TRAIL:(POSITION:32) MYFILE.TST
-/DEF/+2
THIS IS NEW TEXT
/
```

SLP processing generates the following listing file:

```
MYFILE.TST;MYFILE - MYFILE.TST

     1.  ABC
     2.  DEF
     3.  GHI
     4.  KLM
     5.  THIS IS NEW TEXT                    ;**NEW**
     6.  123456789
     7.  456
     8.  789
     9.  CBA
    10.  XYX
    11.  987
```

Again, SLP has numbered the lines in sequence; this time the new input line is inserted two lines beyond the line containing the first occurrence of the string DEF.

3.1.6.4 **Deleting Lines from a File** - The SLP editing command for deleting lines from a file contains two locator fields. Its form is given below.

```
-locator1,locator2[,/audittrail/][;comment]
```

The locator1 and locator2 fields can take any of the forms described in Section 3.1.6.2. The first field, locator1, specifies the line where SLP is to begin deleting lines; locator2 specifies the last line to be deleted. SLP deletes all lines from locator1 through locator2, inclusive.

The example below shows how to delete lines from a file using SLP. The input source file consists of the following lines:

```
ABC
DEF
GHI
KLM
123456789
456
789
CBA
XYX
987
```

The SLP command file for this example is as follows:

```
$EDIT/SLP/LISTING/AUDIT_TRAIL:(POSITION:32)  MYFILE.TST
-/1...9/;/XYX/
/
```

SLP processing generates the following listing file:

```
     1.  ABC
     2.  DEF
     3.  GHI
     4.  KLM
     5.  987                                 ;**-5
```

In this example, the ellipsis (...) is used to abbreviate the larger string 123456789. SLP searches for the first occurrence of the string 1 and the first occurrence of the string 9 on the line, assuming these two strings bracket a larger string, in this case, the string 123456789. SLP begins deleting lines at this line and continues deleting lines until it deletes the last line, specified by the string XYX. SLP applies the audit trail count of the lines it deleted to the next line in the output file.

Using the same input source file, this example shows how to delete a single line using the period locator. The command file for this example is as follows:

```
$EDIT/SLP/LISTING/AUDIT_TRAIL:(POSITION:32) MYFILE.TST
-/DEF/,.
/
```

SLP processing generates the following listing:

```
1.  ABC
2.  GHI                                          $**-1
3.  KLM
4.  123456789
5.  456
6.  789
7.  CBA
8.  XYX
9.  987
```

SLP moves the current line pointer to the line containing the string DEF and then finds the period as the second locator field. Since the second locator field is specified, SLP interprets the editing command as a delete operation and deletes the line containing DEF.

3.1.6.5 **Replacing Lines in a File** - A replacement is a deletion followed by new text. The number of lines deleted need not match the number of lines added. To replace lines in a file, use the full 2-locator command form, as in the delete command. The first line locator field specifies the first line to be deleted. The second line locator field defines the last line in the range to be deleted, which, for replacement operations, is the line where new text is to be inserted.

For example, the command -4,.+4 instructs SLP to move the line pointer to line 4 and replace line 4 and the next four lines (as represented by .+4) with new input lines that immediately follow the command line. This command is equivalent to -4,8.

The example below shows how to delete lines from a file and replace them with new lines. The input source file consists of the following lines:

```
ABC
DEF
GHI
123456789
BCN
CRB
BUR
```

The SLP command file is as follows:

```
$EDIT/SLP/LISTING MYFILE.TST
-2,.+1
NEW LINE 2
NEW LINE 3
/
$EXIT
```

SLP processing generates the following listing file:

```
1.  ABC
2.  NEW LINE 2                                  ;**NEW**
3.  NEW LINE 3                                  ;**NEW**
4.  123456789                                   ;**-2
5.  BCN
6.  CRB
7.  BUR
```

3.1.6.6 **Specifying the Audit Trail Text** - The following SLP edit command changes the text of the audit trail:

```
-,,/newtrail/
```

Here newtrail is the new value (text) of the audit trail. If the length of newtrail exceeds the length specified (or defaulted) for the /AUDIT_TRAIL qualifier, the audit trail is truncated to that length. (The default audit trail, ;**NEW**, is never truncated, even if you specify a length less that 8.)

All subsequent lines added will include the new audit trail text. All lines that indicate where lines have been deleted will include the first character of the new audit trail text as their first character. For example, if you specify the new audit trail JANUARY, the audit trail indicating a replaced line will be J**-2.

When you create a new audit trail, you may want to set the first character of the string to correspond to the comment delimiter that is used in the source file.

3.2 SUMSLP

SUMSLP is a batch-oriented editor similar to the SLP editor. It supplements the functions of SLP by allowing multiple command files to be applied to a single input file. The multiple command files are combined according to fixed rules.

3.2.1 Running SUMSLP

SUMSLP can be run either indirectly from a command procedure or interactively from your terminal. To invoke SUMSLP interactively, you issue a command line in response to the DCL prompt. To invoke SUMSLP from a command procedure, precede the command with a dollar sign ($). The command has the following format:

```
EDIT/SUM[/qualifier(s)]  input-file[/qualifier]
```

/qualifier(s)

A command or file qualifier, as described in Table 3-2. The /OUTPUT and /LIST qualifiers are command qualifiers only; the /UPDATE qualifier is a file qualifier only.

input-file

The file specification for the source file to be edited. File specifications are described above in Section 3.1.2.2.

Table 3-3
SUMSLP Qualifiers

Format	Function
/LIST[=file-spec]	Controls whether a sequence-numbered listing file, showing the original and inserted lines and an audit trail, is produced during the editing process. If you do not specify a file, the listing file takes the same name as the input file, with a file type of LIS. You can specify another file type for the listing file, but LIS is the default. The listing file described in Section 3.2.2.4.
/OUTPUT[=file-spec]	Specifies the output file to be used in the editing operation. If you do not specify a file, the output file has the same name and type as the input file, with a version number one higher than the highest existing version. The output file is described in Section 3.2.2.3.
/UPDATE[=(file-spec,...)]	Indicates the file or files containing the editing commands and changes to be applied to the input source file. If multiple file specifications are listed, they must be separated by commas, and the list must be enclosed in parentheses. The default file type of these files is initially UPD. Default values for the other elements of the file specification are initially taken from the input file specification; after the first file specification in a list, values default to those of the immediately preceding file specification. If no file specification or list of file specifications given, SUMSLP attempts to open a single update file with the same file name as the input file and a file type of UPD. If you do not include the /UPDATE qualifier in the command line, SUMSLP will not attempt to find an update file, but will generate any specified output or listing file. Enter the EDIT/SUM command with the /LIST qualifier but without the /UPDATE qualifier to generate a numbered listing of your source program.

Examples

1. $EDIT/SUM FILE1.MAR/UPDATE

   The input source file FILE1.MAR is updated with the SUMSLP command file FILE1.UPD.

2. $EDIT/SUM FILE2.MAR/UPDATE=UPD2

   The input source file FILE2.MAR is updated with the SUMSLP command file UPD2.UPD.

3. $EDIT/SUM FILE3.MAR/UPDATE=(UPD3A,UPD3B.ENH,UPD3C)

   The input source file FILE3.MAR is updated with the merged contents of SUMSLP command files UPD3A.UPD, UPD3B.ENH, and UPD3C.ENH. The editing commands in the three command files are applied according to the rules given in Section 3.2.3.

3.2.2 SUMSLP Input And Output Files

SUMSLP requires two types of input files: an input source file and one or more SUMSLP command files. SUMSLP produces two types of output files: a source output file and, if requested, a listing file. These four types of files are described in the following sections.

3.2.2.1 **The Input Source File** - The input source file is the file to be updated by SUMSLP. It can contain any number of lines of code.

3.2.2.2 **The SUMSLP Command Files** - SUMSLP command files are very similar to SLP command files, described in Section 3.1.2.2. They need not, however, include an initialization line.

The editing command lines in SUMSLP command files are identical to those used in SLP, with the following exceptions:

- the locator field, described in Section 3.1.6.2, cannot contain strings.
- additional command files cannot be invoked with the at sign (@) operator.

As in SLP, the final editing command line must be followed by a line containing the slash operator (/), which serves as a terminator.

3.2.2.3 **The Output File** - The SUMSLP output file contains the input source file as updated by the additions and changes specified in the SUMSLP command file(s). It does not include an audit trail or line numbers.

If you do not include a file specification for the output file with the /OUTPUT qualifier in the EDIT/SUM command, the output file takes the same file name as the input source file, with a version number one higher than the existing version number.

3.2.2.4 **The Listing File** - The SUMSLP listing file is produced if you specify the /LIST qualifier in the EDIT/SUM command. If you do not specify another name for it, it takes the same file name as the input source file, with the file type of LIS. You can specify another file type, but LIS is the default.

The following example illustrates the generation of a listing file. The input source file, named MYFILE.TST, is:

```
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
TEN
```

There are two SUMSLP command files. The first, UPDATE.UPD, contains the following editing commands:

```
-3,3,/$21-MAR/
INSERTED LINE
/
```

The second SUMSLP command file, NEWLINES.UPD, contains the following editing commands:

```
-7,,/$22-MAR/
NEW LINE
/
```

When the commands in these SUMSLP command files are applied to the input source file, and the /LIST qualifier is applied, the following listing file is produced:

```
                        1 ONE
                        2 TWO
$21-MAR                .1 INSERTED LINE
-1                      4 FOUR
                        5 FIVE
                        6 SIX
                        7 SEVEN
$22-MAR                .1 NEW LINE
                        8 EIGHT
                        9 NINE
                       10 TEN
```

An audit trail is produced automatically unless it has been suppressed (see Section 3.1.5). This field will also contain a marker to indicate the number of lines deleted or replaced from the original file. The marker is placed on the first original line following a deletion and has the form -n, where n is the number of lines deleted.

The line numbers of inserted lines are distinguished from those of original lines by being preceded by a period. Inserted line numbers begin with .1 at the start of each group of new lines.

The source lines show the results of SUMSLP processing.

3.2.3 How SUMSLP Processes Files

SUMSLP applies the edits specified in the SUMSLP command file(s) to the source lines of the input source file. When a list of command files is specified with the /UPDATE qualifier, the editing commands in the various files are arranged according to the following rules:

1. The editing commands are merged into a single stream in ascending order according to the value of locator1 (as described in Section 3.1.6.2). All edits that do not overlap or conflict with any other edits are applied to the source file without any further processing.

2. Editing commands which do conflict are resolved according to the precedence of the SUMSLP command file in which the commands occur. Precedence of SUMSLP command files is determined by the position of the file specifications following /UPDATE. The file specification listed last after /UPDATE has the highest precedence.

   All inserts to the same source line are included in the output file; those from the SUMSLP command file with the highest precedence appear first.

   An operation that deletes or replaces a line will affect not only the specified line, but also any lower precedence inserts or replacements to the same line. A deletion that specifies a range of lines (for example, -10,15) will delete all lines occurring in that range, including inserted lines from SUMSLP command files of lower precedence.

3.3 SLP and SUMSLP MESSAGES

The following sections describe the diagnostic messages issued by SLP and SUMSLP.

3.3.1 SLP Messages

The following sections describe the information and error messages issued by SLP. Each message is followed by an explanation of its meaning and a recommendation for user action.

3.3.1.1 SLP Information Messages

SLP -- COMMAND FILE CHECKSUM IS ######

Explanation: By specifying the /CHECKSUM qualifier in the command line, you requested SLP to calculate the checksum value for the edit commands.

User Action: This message is for your information only. No action is required.

SLP -- *DIAG*-ERROR IN COMMAND FILE filespec CHECKSUM

Explanation: An incorrect value was specified for the command file checksum. If you enter the edit command lines directly from the terminal, the command file in the error message is CMI.CMD. Thus, the error message reads:

SLP -- *DIAG* - ERROR IN COMMAND FILE CMI.CMD CHECKSUM

User Action: This is a warning message only. The specified output file is still created, although possibly not as intended.

SLP -- *DIAG*-n LINES TRUNCATED BY AUDIT TRAIL
command line

Explanation: Line truncation by the audit trail was detected.

User Action: This message is for your information only. The specified output file is still created. (In the listing file, a question mark (?) replaces the period (.) in the line number of the lines that were truncated. It is possible that audit-trail strings from the input file will be truncated by the new audit-trail string although text strings will not be truncated.) Determine where the truncation(s) occurred. If necessary, modify the command file so that it contains commands that do not cause truncation.

3.3.1.2 **SLP Error Messages** - The SLP error messages listed below are issued in two formats:

- SLP followed by two dashes, the type of error message, and the error message. If applicable, the command line or command line segment that caused the message is printed on the next line. For example:

  ```
  SLP -- *FATAL*-ILLEGAL SWITCH
  $EDIT/SLP/TUNCATE
  ```

- SLP followed by two dashes, the type of error message, the error message, and the name of the file with which the error is associated. For example:

  ```
  SLP -- *FATAL*-OPEN FAILURE LINE LISTING FILE filename
  ```

SLP -- *FATAL*-COMMAND SYNTAX ERROR
command line

Explanation: The command line format did not conform to syntax rules. Open files were closed and SLP was reinitialized.

User Action: Reenter the command line.

SLP -- *FATAL*-ILLEGAL DEVICE NAME
command line

Explanation: The device specified was not a legal device. Open files were closed and SLP was reinitialized.

User Action: Reenter the command line.

SLP -- *FATAL*-ILLEGAL DIRECTORY
command line segment

Explanation: The directory was not specified correctly. Open files were closed and SLP was reinitialized.

User Action: Reenter the command line with a correctly specified directory.

SLP -- *FATAL*-ILLEGAL ERROR/SEVERITY CODE p1 p2 p3

Explanation: This error message indicates that an error occurred in the SLP program.

User Action: Reenter the command line. If the error persists, submit a Software Performance Report (SPR) along with the related console dialogue and any other related information, such as programs or listings.

SLP -- *FATAL*-ILLEGAL FILE NAME
command line segment

Explanation: A file specification was longer than 30(8) characters or contained a wild card character (that is, an asterisk in place of a file specification element). Open files were closed and SLP was reinitialized.

User Action: Reenter the command line.

SLP -- *FATAL*-ILLEGAL GET COMMAND LINE ERROR

Explanation: The system was unable to read a command line. This indicates an internal system failure or an error in the SLP program.

User Action: Reenter the command line. If the error persists, submit a Software Performance Report (SPR) along with the related console dialogue and any other pertinent information.

SLP -- *FATAL*-ILLEGAL SWITCH
command line segment

Explanation: Either the qualifier used was not a valid SLP qualifier or a legal qualifier was used in an invalid manner. Open files were closed and SLP was reinitialized.

User Action: Reenter the command line with the correct qualifier specified.

SLP -- *FATAL*-INDIRECT COMMAND SYNTAX ERROR
command line

Explanation: The command line format specified for the SLP command file did not conform to syntax rules. Open files were closed and SLP was reinitialized.

User Action: Reenter the command line.

SLP -- *FATAL*-INDIRECT FILE DEPTH EXCEEDED
command line

Explanation: More than three levels of indirection were specified in a SLP command file. Open files were closed and SLP was reinitialized.

User Action: Correct the command file and reenter the command line.

SLP -- *FATAL*-I/O ERROR COMMAND INPUT FILE

or

SLP -- *FATAL*-I/O ERROR COMMAND OUTPUT FILE

or

SLP -- *FATAL*-I/O ERROR CORRECTION INPUT FILE filename

or

SLP -- *FATAL*-I/O ERROR LINE LISTING FILE filename

or

SLP -- *FATAL*-I/O ERROR SOURCE OUTPUT FILE filename

Explanation: One of the following conditions may exist:

- A problem exists on the physical device (for example, the disk is not spinning).
- The length of the command line was greater than the specified number of characters.
- The file is corrupted or the format is incorrect.

User Action: Determine which condition caused the message and correct that condition. Reenter the command line.

SLP -- *FATAL*-INDIRECT FILE OPEN FAILURE
command line

or

SLP -- *FATAL*-OPEN FAILURE CORRECTION INPUT FILE filename

or

SLP -- *FATAL*-OPEN FAILURE LINE LISTING FILE filename

or

SLP -- *FATAL*-OPEN FAILURE SOURCE OUTPUT FILE filename

Explanation: One of the following conditions may exist:

- The file is protected against an access.
- A problem exists with the physical device (for example, the device was not online).
- The volume is not mounted.
- The specified file directory does not exist.
- The named file does not exist in the specified directory.

These errors cause open files to be closed and SLP to be reinitialized.

User Action: Determine which condition caused the message and correct that condition. Reenter the command line.

SLP -- *FATAL*-LINE NUMBER ERROR
command line

Explanation: The command line printed contained an illegally-specified numeric line locator.

User Action: Terminate the SLP edit session and refer to the rules for specifying numeric line locators in Section 3.1.6.2. Correct the error and reenter the command line.

SLP -- *FATAL*-PREMATURE EOF CORRECTION INPUT FILE filename

Explanation: An out-of-range line locator was specified in a correction file or from the terminal; for example, -1000 was specified for an 800-line file.

User Action:

- Terminate the current editing session.
- Restart the editing session, entering the correct line number.

SLP -- *FATAL*-PREMATURE EOF COMMAND INPUT FILE

Explanation: This error occurs if you do not terminate SLP command input with a slash (/) or if you inadvertently type <CTRL/Z> at the terminal, which sends an end-of-file to SLP before the slash (/) character is read. SLP prints SLP>, indicating that a new file specification is expected.

User Action: Restart the editing session at the point where the CTRL/Z was typed.

3.3.2 SUMSLP Messages

The following sections described the information and error messages issued by SUMSLP. All of the error messages are warnings. Each message is followed by an explanation of its meaning and, where appropriate, a rrecommendation for user action.

3.3.2.1 SUMSLP Information Message

SUM-I-EDIT$CLSH, edits clash
editing commands files specifications

Explanation: Two or more conflicting editing commands have been entered, that is, more than one edit operation has been specified for one line of source code. The relevant editing commands and the file specifications of the SUMSLP command files are listed following the message.

User Action: This message is for your information only. No action is required.

3.3.2.2 SUMSLP Error Messages

SUM-W-EDOUTSEQ, edits out of sequence

Explanation: The editing commands from a single SUMSLP command file were nòt in ascending sequence. The edits have been moved to the correct position.

User Action: Check output file. It should not be necessary for you to edit the file again.

SUM-W-PRMEOF, premature end-of-file

Explanation: A SUMSLP command file has terminated unexpectedly. This is probably due to the absence of a terminator (/) at the end of the command file.

User Action: Insert a terminator (/) at the end of the command file and edit again.

SUM-W-SLPSYNERR, SLP command syntax error
editing command file specification

Explanation: The editing command did not conform to the SUMSLP syntax rules. However, processing of edits continued.

User Action: Check output file and, if necessary, edit again with corrected editing commands.

CHAPTER 4

DISK SAVE AND COMPRESS UTILITIES

The Disk Save and Compress (DSC) utilities are used to back up and restore disk volumes that have been formatted and initialized as Files-11 structure Level 1 or Structure Level 2 volumes. There are three DSC utilities:

- DSC2 saves, compresses, and restores Files-11 Structure Level 2 disk volumes. DSC2 runs online under the control of the VAX/VMS operating system.
- DSC1 performs the same functions as DSC2 for Files-11 Structure Level 1 disk volumes. DSC1 runs online under the control of the VAX/VMS operating system.
- DSC-2 is a stand-alone version of the online utility DSC2. DSC-2 is a component of the VAX-11 Diagnostic Package, and is bootstrapped from diagnostic floppy diskettes.

This chapter describes the uses and features of the DSC utilities; it is organized into four sections:

- Section 4.1, "Typical Uses of DSC," introduces the most common uses of the DSC utilities.
- Section 4.2, "Specifying DSC Commands," defines the DSC command line format and the functions of the DSC file qualifiers.
- Section 4.3, "Using the DSC Utilities," demonstrates, with examples, some of the typical uses for the DSC utilities.
- Section 4.4, "Auxiliary Procedures for DSC Operations," describes procedures useful to DSC users.
- Section 4.5, "DSC Messages and Error Recovery Procedures," defines all DSC-generated messages and indicates how you can recover from DSC-related error conditions.

To use the DSC utilities, you should be a VAX/VMS operator and be familiar with the following manuals:

- VAX-11 Software Installation Guide
- VAX/VMS Operator's Guide
- VAX/VMS Command Language User's Guide

You should also be familiar with BAD and VFY, volume maintenance utilities described in this manual.

For information about Files-11 Structure Level 1 and Structure Level 2 disk volumes, refer to:

- VAX/VMS System Manager's Guide, which describes the Files-11 disk structures in the context of backing up public disk volumes.
- Introduction to VAX-11 Record Management Services, which describes the Files-11 disk structures in the context of general disk organization
- VAX-11 Record Management Services Reference Manual, which explains RMS-related record, file, and volume concepts and formats.

4.1 TYPICAL USES FOR DSC UTILITIES

You use the DSC programs to back up and restore entire disk volumes including the system files that define volume structure. The disk volumes can be single-device volumes or disk volume sets (which consist of one or more single-device disks bound together).

One typical use of the DSC utilities is restoring volumes which were backed up. The purpose for using a DSC program to back up a disk volume is to save a copy of the volume in case the data on the original volume is corrupted. When you back up a disk volume onto another disk, you create a usable copy of the original disk. When you back up a disk volume onto a tape set, you create a file which can later be used to restore the original disk.

Another typical use of a DSC utility is copying the VAX/VMS distribution medium. This operation, which uses stand-alone DSC-2, is described in the VAX-11 Software Installation Guide.

Other typical uses for the DSC utilities, as described in the following sections, are:

- Backing up the VAX/VMS system disk
- Backing up public or private disk volumes
- Compressing the files on a public or private disk volume
- Regulating disk bad-block information
- Comparing the contents of two volumes
- Transporting volumes

4.1.1 Backing Up the VAX/VMS System Disk

Use the stand-alone DSC-2 program to back up a VAX/VMS system disk.

Immediately after installing a new version of VAX/VMS, whether from a DSC copy of the distribution medium or from a VAX/VMS update kit, back up the new system using DSC-2 before you bring up the installation for general time-sharing usage.

The system manager should provide schedules that define when the VAX/VMS operating system should be shut down to back up the system disk. On these occasions, you follow the shut-down procedures in the VAX/VMS Operator's Guide and then use the DSC-2 program to back up the system disk.

Although you could use the online DSC2 program to back up a system disk (executing the procedure from any user terminal), DIGITAL recommends that back-ups of the system disk be done when the system is shut down. You execute the procedure using DSC-2 from the system console.

Note that operators at installations that have only one disk drive must use the stand-alone DSC-2 program to back up the disk (onto magnetic tape). Refer to the VAX/VMS Operator's Guide for instructions.

4.1.2 Backing Up Public or Private Disk Volumes

To back up public or private disk volumes onto disk, refer to the VAX/VMS Operator's Guide. Refer to the examples in this chapter if you are requested to copy the contents of the volume(s) to magnetic tape.

Use the online DSC programs DSC2 (for Files-11 Structure Level 2 volumes) or DSC1 (for Files-11 Structure Level 1 volumes) to back up public and private disk volumes.

4.1.3 Compressing the Files on a Public or Private Disk Volume

To compress files on a disk volume (if backing up to tape) you back up the volume and then immediately restore it. Because of the way DSC programs execute, you cannot back up a volume without that volume being compressed. How much compression occurs depends upon how the volume has been used. For example, there may be very little compression on a volume that has been used as a private volume; but there may be a great deal of compression on a volume that has become fragmented through general use.

4.1.4 Regulating Disk Bad Block Information

You may need to establish and keep current the bad block information on the disk devices at your installation.

Bad block information is established for a disk when it is initialized as a Files-11 volume. Section 5.2 describes how the DCL command INITIALIZE command formats and labels a disk as a Files-11 volume; refer to the VAX/VMS Command Language User's Guide for a complete description of the INITIALIZE command.

As a disk device is used, the probability of a disk data error increases. To ensure that only good data blocks are allocated to users of a disk, you may need to establish a procedure that revises the bad block information on a disk device. One such procedure could be:

1. Run a DSC program to back up a disk volume, either to magnetic tape or to another disk (that contains up-to-date bad block information).

2. Run the Bad Block Locator (BAD) utility program (described in Chapter 5) to determine the number and location of bad blocks on the disk which you have just backed up. You supplement the bad block information on the disk with the bad blocks located by BAD.

3. Run the DSC program again to restore the original volume.

4.1.5 Comparing the Contents of Two Volumes

You can use a DSC program to compare the contents of two disk volumes or the contents of a disk volume and a tape set. For example, suppose you are using DSC to make 10 copies of a disk volume. You could execute the DSC program 10 times, each time copying from the original volume to a new volume, and each time specifying the Verify qualifier to ensure that the volumes are identical. (Refer to Section 4.3.3 for information on the use of the Verify qualifier.)

Or, you could save time by using the DSC Compare qualifier. The procedure would be:

1. Back up the original volume using DSC with the Verify qualifier.

2. Using the output volume from the previous DSC operation each time as the input volume to the next operation, execute DSC nine more times, without the Verify qualifier.

3. Finally, use DSC with the Compare qualifier to compare the contents of the first output volume and the last output volume. If the volumes compare successfully, all 10 copies are good; if the volumes do not compare, the operations must be repeated.

You must decide whether the time saved by using this procedure is worth the (slight) risk of complementary errors being carried throughout the operations.

NOTE

Because the DSC programs do not support tape-to-tape transfers, they cannot be used to compare tape sets.

4.1.6 Transporting Volumes

The DSC1 program provides a way to transport the contents of Files-11 Structure Level 1 volumes between VAX/VMS installations and between VAX/VMS and RSX-11M, RSX-11M-PLUS, and IAS installations. For

example, you may need to deliver the contents of several disk volumes to another system. If it is impractical to physically deliver the disk volumes, you can use the DSC1 program to save the volumes on tape sets and deliver the tape sets to the other installation. The operator at that installation can then restore the volumes from the tape sets using the DSC1 program.

Similarly, the DSC-2 and DSC2 programs can be used to transport Files-11 Structure Level 2 volumes between VAX/VMS systems.

4.1.7 Device Transfers Supported by DSC Programs

The DSC programs support single- and multivolume file transfers between disk devices and between disk and 9-track magnetic tape devices. DSC programs do not support transfers between magnetic tape devices. All devices supported by VAX/VMS, as listed in Appendix A, are supported by the DSC programs.

Note an important distinction between disk-to-disk transfers and disk-to-tape transfers. When a DSC program is used to back up a disk to another disk, that output disk is a usable disk volume. However, a tape set produced by a DSC program has no use, except to be input to another DSC operation which is used to restore a disk from that tape set.

4.2 SPECIFYING DSC COMMANDS

This section describes how to invoke and terminate the DSC programs, enter DSC command strings, and define DSC operations.

4.2.1 Invoking and Terminating Online DSC1 and DSC2

To invoke DSC1 or DSC2, type one of the following commands in response to the DCL prompt.

For DSC1:

```
$ RUN SYS$SYSTEM:DSC1
```

For DSC2:

```
$ RUN SYS$SYSTEM:DSC2
```

The utility will reply with a prompt, indicating that it is ready to accept a command string. The prompts are DSC> and DSC2>. You enter a command string (followed by a carriage return) for the operation you want performed. If the operation is successful, no completion message will be displayed, and the utility will prompt for another command string.

Terminate DSC by typing <CTRL/Z> in response to the DSC prompt.

4.2.2 Invoking and Terminating Stand-alone DSC-2

The stand-alone DSC-2 program does not run under control of the VAX/VMS operating system. It is a component in the VAX-11 Diagnostic Package, and is distributed on floppy diskettes with the package.

To invoke DSC-2, follow the procedure "Backing Up Tape and Disk Volumes," described in the VAX/VMS Operator's Guide. The DSC-2 prompt is identical to the DSC2 prompt:

```
DSC2>
```

To terminate DSC-2, follow the procedure described in the VAX/VMS Operator's Guide.

4.2.3 Specifying the DSC Command String

The DSC command string specifies the operation that the DSC program is to perform. The command string format is identical for the three DSC programs. Note that the left side of the equal sign in this format denotes output parameters, and the right side denotes input parameters.

```
devcu:  [,devcu:...] [dsclabel] [/qualifier(s)]=

                devcu:  [,devcu:...] [dsclabel] [/RW]
```

devcu

The physical device(s) to or from which data is to be transferred, where dev is the 2-character device code, c is the device controller letter, and u is the device unit number, followed by a colon (:). For example:

```
MTA0:
```

If there is more than one output device, specify each and separate them with commas. For example:

```
MTA0:,MTA1:,MTB3:
```

dsclabel

An optional file label for the DSC file created when a disk-to-magnetic tape operation is performed. If specified, the label must be a 1- to 12-character alphanumeric label. If the label is not specified in the output specification (disk-to-tape operation), the volume label of the input disk volume becomes the output DSC file label. If the label is not specified in the input specification (tape-to-disk operation), the DSC programs begin the operation with the first file on the first input device in the command string.

For disk-to-disk operations, the label you specify becomes the volume label of the output disk. If not specified, the output disk volume label remains the same as the input disk volume label.

Specify the label before any qualifiers.

/qualifier(s)

The output file qualifiers determine the operations to be performed. They are defined in Table 4-1. They can be specified in any order; for example, /DENS=1600/VE has the same effect as /VE/DENS=1600.

/RW

The only valid input qualifier is /RW. When you specify /RW, the DSC program first rewinds the tape reel mounted on the first input device in the command string and then transfers data from it.

Table 4-1
DSC Output File Qualifiers

Format	Name and Function
/AP	The Append qualifier appends files to the tape volume specified by the first output device in the command string. Use /AP with output tapes only. /AP is required if you want to preserve the information on the output tape and the tape is either at beginning-of-tape or you have also specified the /RW output qualifier.
/BAD=MAN	The Add Bad Blocks qualifier allows you to supplement the output disk volume's bad block file with manually entered bad blocks.
/BAD=NOAUTO	The Ignore Bad Block File qualifier allows you to ignore the bad block file on the output disk for this operation. Using this qualifier results in an allocated, but empty, bad block file on the output disk volume.
/BAD=MAN:NOAUTO	The Replace Bad Block File qualifier allows you to replace the output disk's bad block file with manually entered bad blocks during the DSC operation.
/CMP	The Compare qualifier does not produce data transfers. When you use /CMP, you are comparing the contents of the input device with the contents of the output device, and identifying any differences that may be present.
/DENS=1600	The Density qualifier allows you to create output tape volumes recorded at 1600 bits per inch (bpi). If not specified, output tapes are recorded at 800 bpi.
/RW	The Rewind qualifier causes the output tape volume to be rewound before data is written to it. Any data on the output tape is overwritten unless the /AP qualifier is also specified.
/VE	The Verify qualifier causes the DSC program to compare the contents of the output and input volumes and identify differences between them. The verify operation occurs after the data transfer has completed.

4.3 USING DSC PROGRAMS

This section demonstrates common uses of the DSC programs. Topics include:

- Setting up for DSC operations
- Using DSC file labels
- Using DSC qualifiers

4.3.1 Setting Up for DSC Operations

There are several factors to consider when setting up for DSC operations:

- User privileges required. If you are not the owner of a volume that is to be backed up, you need the VOLPRO privilege to mount the volume using the MOUNT/FOREIGN command.
- Foreign mounting of output devices. All DSC output devices (both disk and tape) must be mounted with the DCL command MOUNT using the /FOREIGN qualifier.
- Valid scratch devices. Ensure that all output devices are valid scratch devices. Remember that most DSC operations destroy the original contents of the output volumes.
- Physical device names. Use only physical device names in input and output device specifications. Using logical names, although valid, increases the risk of inadvertently specifying an input volume as an output device. DSC programs initialize output disks before transferring data to them.
- Placed files. Since the DSC programs do not support files created with placement controls, do not execute DSC to save or compress disks unless you and the volume's owner agree that file placement controls can be destroyed. For information about file placement controls, refer to the RMS-11 User's Guide or the VAX-11 Record Management Services Reference Manual.
- Index file placement. When you use the INITIALIZE command to initialize a disk volume, you can specify the placement of the index file for the volume's directory structure by means of the /INDEX qualifier. (See the VAX/VMS Command Language User's Guide for information on INITIALIZE.) However, when you use DSC to initialize an output disk, it always places the index file at the beginning of a disk volume.
- Allocation errors when backing up a system disk. An allocation error that occurs during a back-up of the system disk usually indicates that the output disk contains too many bad blocks and does not have enough contiguous space for some of the larger contiguous system files. Retry the operation, using another output disk.
- Single disk systems. If your installation is configured with only one disk drive, you must use the stand-alone DSC-2 program to perform the operations described in this chapter. Once in the stand-alone environment provided by DSC-2, you can perform all of the online DSC2 functions described in this chapter.

4.3.2 Using File Labels

For disk-to-tape operations, use DSC file labels to identify tape volumes as DSC-produced files.

If you do not specify an output file label, the DSC programs use the input disk volume label to construct DSC file label(s) for the output tape volume(s). When you specify an output DSC file label, the DSC programs construct output tape file labels from the label you provide.

If you do not specify an input file label during a restore operation from tape, the DSC programs transfer the first DSC-produced file from the first input tape device in the command string. When you specify an input file label during a restore operation from tape, the DSC programs search for that file and begin the operation, if the file is found. Otherwise, the following error message is generated:

```
DSC -- *FATAL* 75 TAPE FILE file label NOT FOUND
```

For disk-to-disk operations, you may optionally use a DSC file label to create a new volume label for the output disk.

Example: Saving a Private Volume Using DSC2

You are asked to save an RP06 volume with the disk volume label FORTVOLV3 on tape. The DSC file label for the back-up tape set is to be FORTBACK1. The back-up is expected to consume two full reels of tape.

Mount the input disk on DBA1 and the output tapes on MTA1 and MTA2. Invoke DSC2 and then enter the following command string in response to the DSC prompt:

```
DSC2>MTA1:,MTA2:FORTBACK1=DBA1:
```

This operation results in an output tape set containing the multivolume file FORTBACK1.

Had you not specified the output file label, DSC2 would have used the input disk's volume label to construct the output file label, which would then be FORTVOLV3.

Example: Restoring a Private Volume Using DSC2

To restore the volume containing the file labeled FORTBACK1 onto an RP06, mount the input tape set, FORTBACK1, on MTA1 and MTA2 and the RP06 on DBA3. Invoke DSC2 and then enter the following command string in response to the DSC prompt:

```
DSC2>DBA3:=MTA1:,MTA2:FORTBACK1
```

This operation results in a restored RP06 on DBA3 with the disk volume label FORTVOLV3. Because you specified an input file label, DSC2 searched for that file before beginning the data transfers to the output disk. Had you not specified an input file label, DSC2 would have attempted to transfer the first DSC-produced file it found on MTA1.

In this case, FORTBACK1 was the only DSC-produced file on the tape. If there were more than one DSC-produced tape file on the volume, however, you would risk copying the wrong file to the output disk by not specifying a file label.

4.3.3 Using the Verify Qualifier

Use the Verify (/VE) output qualifier when you want DSC to save, then verify the copy in the same operation. Specifying /VE directs the DSC program to compare each logical block on the input and output volumes and to identify blocks that do not compare.

Note the difference between the use of the Compare qualifier (refer to Section 4.3.7) and the Verify qualifier. DSC operations involving the Compare qualifier produce no data transfers, but an operation involving the Verify qualifier results in an output volume, then a verification of that volume.

An example of using the stand-alone DSC-2 program to save and verify a system disk is included in the VAX/VMS Operator's Guide.

Example: Backing Up and Verifying a Disk Volume Using DSC2

You are asked to back up an RP06 disk onto a tape, verifying the copy and using the input disk's volume label (FORTDATA1) as the output tape file label.

Mount the RP06 on DBB5 and the output scratch tape on MTA2. Invoke DSC2 and them enter the following command string in response to the DSC prompt:

```
DSC2>MTA1:/VE=DBB5:
```

The copy operation is completed before the verify operation begins. When the copy is completed, the output tape is rewound, and DSC2 issues the following message as it begins the verify operation:

```
DSC -- 45 STARTING VERIFY PASS
```

As the verify operation proceeds, warning messages are issued if any logical blocks do not compare. The messages are similar to the following example:

```
DSC -- *WARNING* VERIFICATION ERROR ON DBB5:
FILE ID 000623,000004,000000,VBN 000000,000001
```

Note that the output tape is not rewound at the end of the DSC operation unless more than a single volume (tape) is required. In that case, all tape volumes will be rewound and unloaded when the verification is complete.

Example: Backing Up and Verifying a Disk Volume Set Using DSC2

You are asked to back up onto magnetic tape and to verify the copy of a disk volume set consisting of two RK07 disk volumes, mounted on DMA1 and DMA2. The volume set label is MASTER_SET. You have one tape drive, MTA1, available. The backup will consume two reels of tape.

Invoke DSC2 and enter the following command string to back up the first volume in the set:

```
DSC2>MTA1:MASTER_SET/VE=DMA1:
```

The result of this operation is that the contents of DMA1 are copied to the tape on MTA1 until that reel is full. Then the tape is rewound, and the copy (to that point) is verified. The tape is then rewound and unloaded. DSC2 issues the following message:

```
DSC -- 44 MOUNT REEL 2 ON MTA1:  AND HIT RETURN
```

At this point, remove the first backup tape and mount another on MTA1. When you press carriage return, DSC resumes the backup operation and issues the following message:

```
DSC -- 46 RESUME COPYING
```

The copy operation continues until the rest of DMA1 has been transferred. Then the verify operation resumes. DSC2 issues the message:

```
DSC -- 45 STARTING VERIFY PASS
```

DSC2 then verifies the rest of DMA1. At the end of the operation, the tape on MTA1 is rewound and unloaded. Mount a new tape on MTA1 and enter another command string to back up the volume that is mounted on DMA2:

```
DSC2>MTA1:MASTER_SET/VE=DMA2:
```

This second operation transfers DMA2 to magnetic tape.

4.3.4 Using the Density Qualifier

Use the Density (/DENS=1600) output qualifier when you want output tape sets to be recorded at 1600 bpi. If you do not specify the Density qualifier, output tapes are recorded at the default density of 800 bpi.

Note that the DSC programs automatically read DSC-produced tapes at the recorded density when restoring disks from tape sets.

Example: Backing Up a System Disk to Tape at 1600 bpi

Using stand-alone DSC-2, you are to back up the system disk, an RP06 with the volume label MYVMSRL1, onto a tape set recorded at 1600 bpi. The DSC file label for the tape set is to be MONDAY; the tape drive available for the back-up is MTA2.

Load the stand-alone DSC-2 program from the floppy diskette. Mount the RP06 on device DBA0 and the scratch reel on MTA2. To perform the operation, enter the following command line in response to the DSC-2 prompt:

```
DSC2>MTA2:MONDAY/DENS=1600=DBA0:
```

The result of this operation is an output tape set with the DSC file label MONDAY, recorded at 1600 bpi. You would specify this file label in a DSC-2 operation to restore the system disk from the tape set.

4.3.5 Using the Rewind Qualifier

Use the Rewind (/RW) qualifier as an input qualifier when you want to restore a disk from a tape set. Typically, you specify /RW when:

- You are certain that the first DSC-produced file on the first mounted input tape is the correct file.
- You have specified an input DSC file label in the command string and you want the first input tape rewound and searched for that file label before the data transfer occurs

Use the Rewind qualifier as an output qualifier when you want to save a disk onto a tape set. Typically, you would specify /RW when you want output tapes rewound to beginning-of-tape before any data transfers occur.

Example: Backing Up an RP06 Volume onto a Tape Set Using DSC-1

You are asked to back up an RP06 disk that is a Files-11 Structure Level 1 volume. THE RP06 is on DBA6, and two tape drives, MTA1 and MTA2, are available for the backup. The DSC file label for the output tape set is to be MRPMON.

Invoke DSC1 and enter the following command string in response to the DSC prompt:

```
DSC>MTA1:,MTA2:MRPMON/RW=DBA6:
```

The result of this operation is a copy of the RP06 on as many tape reels as are necessary to contain the volume. The tape set created has the DSC file label MRPMON. Because you specified /RW, the first tape mounted on MTA2 is rewound before data is transferred to it. Subsequent tapes mounted on MTA2 are also rewound before data is transferred.

Example: Restoring an RP06 Volume from a Tape Set

To restore the RP06 in the example above from the tape set, mount the RP06 on device DBB2 and the first and second relative volumes of the DSC-produced file labeled MRPMON on MTA2 and MTA1. Enter the following command string:

```
DSC>DBB2:=MTA2:,MTA1:MRPMON/RW
```

The result of this operation is a restored RP06 on device DBB2. The DSC1 program checks for the correct DSC-produced file label and volume sequence number after rewinding the tape on MTA2 and starts the transfer. The restore operation continues between tapes on MTA2 and MTA1. When the contents of one tape are transferred, it is rewound and unloaded as the transfer continues from the next tape. When the last tape volume has been transferred, it is rewound and unloaded. As the copying proceeds from reel to reel, DSC checks the volume sequence number to ensure that the tapes are mounted in the correct order.

4.3.6 Using the Append Qualifier

Use the Append (/AP) output qualifier to add a DSC file to a tape volume that already contains other DSC-produced files. Typically, the Append qualifier is used to copy the contents of several small disk volumes to a tape set. Note that you use /AP only when appending separate DSC-produced files to an output tape set. Also, you can append DSC files to only the first volume in an output tape set.

Example: Backing Up Three Separate Disk Volumes Using DSC1

You need to back up onto tape three RK07 disk volumes with volume labels FORTVOL01, FORTVOL02, and FORTVOL03.

The first RK07 is mounted on the only available disk device, DMA1, and scratch tapes are mounted on devices MTA1 and MTA2.

Invoke DSC1 and start the first of three operations required by entering the following command string in response to the DSC prompt:

```
DSC>MTA1:,MTA2:/RW=DMA1:
```

The result of this operation is a copy of FORTVOL01 to the tape on MTA1, with the output DSC file label FORTVOL01. The output tape has been rewound. At this time, physically dismount the first RK07 and replace it with the second RK07. To begin the second operation type the following:

```
DSC>MTA1:,MTA2:/AP=DMA1:
```

The result of this operation is a transfer of the second RK07 volume to the tape on MTA1. If the transfer did not fill the reel, you can append FORTVOL03 to that reel, by following the same procedure. If necessary, the third volume can continue onto MTA2, and onto as many other tape volumes as necessary to complete the transfer.

If the transfer did fill the reel, that reel would be rewound and unloaded, and the transfer of FORTVOL02 would continue on MTA2. In this situation, you cannot append FORTVOL03 to the tape set, because an Append qualifier can be used only to append files to the first output volume in a tape set.

4.3.7 Using the Compare Qualifier

Use the Compare (/CMP) output qualifier to compare the contents of two single-volume disks or a single-volume disk and a DSC-produced tape set.

Example: Comparing a Disk to a Tape Set Using DSC2

You have been asked to use DSC2 to compare the contents of a 2-volume tape set which has the DSC label PAYBACKUP with the RP06 disk volume labeled PAYMASTER.

Mount the RP06 on device DBA2 and the tape volumes on MTA2 and MTA3. Invoke DSC2 and enter the following command string in response to the DSC prompt:

```
DSC2>DBA2:/CMP=MTA2:,MTA3:PAYBACKUP/RW
```

The result of this operation is a listing of the blocks that do not compare. For each block that did not compare, a message similar to the following is issued:

```
DSC -- *WARNING* 42 VERIFICATION ERROR ON DBA2:
FILE ID 000221,000050,000000,VBN 000026,000013
```

This operation produces no data transfers: the contents of both input and output volumes are unchanged. Thus, the same results could have been obtained had you executed the operation by typing:

```
DSC2>MTA2:,MTA3:PAYBACKUP/RW/CMP=DBA2:
```

NOTE

When comparing a tape set with a disk which was not the input disk when the tape was created, it is recommended that the tape be specified as input.

4.3.8 Using Bad Block Qualifiers

On occasion you may need to use a DSC program to regulate the contents of a disk volume's bad block file before you restore that disk. On most disks, as described in Section 5.1.2, the last several blocks contain a description of the bad blocks on the disk. This description is left by the formatter or by the BAD utility. By default, DSC constructs the volume's bad block file (BADBLK.SYS) from this data. The bad block file includes the bad blocks and the bad block description.

The DSC programs provide three output qualifiers for bad block manipulation.

Qualifier	Function
/BAD=MAN	Adds logical block numbers to the disk's bad block file
/BAD=NOAUTO	Ignores the disk's bad block information for the following DSC operation
/BAD=MAN:NOAUTO	Replaces the disk's bad block information with the logical block numbers entered during the operation

NOTE

General procedures for establishing and maintaining the disk bad block file are described in the VAX/VMS Operator's Guide. That manual defines the situations in which you use other methods for manipulating the contents of disk volume bad block files. See also the Bad Block Locator Utility (BAD), described in Chapter 5, below.

4.3.8.1 **Using the /BAD=MAN Qualifier** - If you include the /BAD=MAN qualifier in the DSC command string, the utility will prompt you for the logical block numbers of the bad blocks that you want to add to the volume's bad block file.

```
DSC>BAD=
```

You can enter the bad block numbers individually, pressing carriage return after each entry, or you can enter the bad block numbers in groups. For example:

```
DSC>BAD=702.:3

DSC>BAD=621.,622.

DSC>BAD=4057.
```

The first command enters bad blocks numbers (in decimal) 702, 703, and 704; the second command enter blocks 621 and 622; the third command enters block 4057.

To terminate the list, press return in response to the prompt DSC>BAD=.

The bad blocks entered manually become part of the disk's bad block file; however, they do not become part of the permanently recorded bad block description. This means that if the disk is later written onto by DSC, the bad blocks that you have previously entered will be forgotten.

4.3.8.2 **Using the /BAD=NOAUTO Qualifier** - Disk bad block files prevent data corruption by not allowing the blocks marked as bad to be written into by user programs. However, there may be occasions which force you to ignore the bad block file on an output disk in order to complete a DSC operation. For example, it may be impossible to allocate a very large contiguous file on the output disk, due to bad blocks. On some disks, such as the RP06, it is possible to use the bad blocks and rely on ECC correction. (This technique is not advised where high data reliability is of paramount importance.)

Enter the following command string:

```
DSC2>DBA2:/BAD=NOAUTO=DBA1:
```

4.3.8.3 **Using the /BAD=MAN:NOAUTO Qualifier** - To replace a disk's bad block file with a new list of bad blocks, include the /BAD=MAN:NOAUTO qualifier in the DSC command string, as in the following example:

```
DSC2>DBA5:/BAD=MAN:NOAUTO=DBA6:
```

DSC2 prompts for the bad block numbers:

```
DSC>BAD=
```

You respond with the appropriate numbers, as shown in Section 4.3.9.1.

After the list is terminated by your responding with a <RET> to the DSC prompt, the bad blocks that you entered replace the bad blocks in the bad block file. Then disk-to-disk transfer begins.

4.4 AUXILIARY PROCEDURES FOR DSC OPERATIONS

Two procedures are useful to DSC users:

- Translating file identifications into file specifications
- Converting physical disk addresses to logical block numbers

4.4.1 Translating File Identifications into File Specifications

Many of the DSC messages report the file identification of a disk volume's file. While a file identification is useful to identify a unique file in a disk volume (or disk volume set), it does not tell you who is the owner of the file; what is the owner's UIC; what is the file name, file type, and version number; or what protection codes are associated with the file.

You may need to translate a file identification from its virtual block number format to a more readable form. You can do this using the MCR DUMP command.

For example, DSC-2 may issue the following message during a back-up operation:

```
DSC -- *WARNING* 41 I/O INPUT ERROR I ON DB1: FILE ID 000050,000002
```

This back up operation continues to completion. To find out more information about the deleted file (it was not copied to DBA2), use the following procedure:

1. Reboot the VAX/VMS operating system here on DBA1).

2. Enter the following command string in response to the DCL prompt:

   ```
   $ MCR DMP TI:=DBA1:/FI:50:2/HD/BL:0
   ```

   The file header (FILE ID 000050,000002, given in the warning message above) is printed at the terminal.

From the record of this display, you can identify the file specification and judge whether the deletion was appropriate. If not, you may need to repeat the back-up or take some other action to restore the file.

4.4.2 Converting Disk Addresses to Logical Block Numbers

Some disk manufacturers provide a list of bad blocks with a disk device when it is shipped from the factory. Some disk diagnostic programs (including the program ESRAC in the VAX-11 Diagnostics Package) can be run to generate a list of bad blocks on a disk. Often, these lists identify bad blocks by cylinder, track, and sector number.

If you want to use a DSC program to enter these bad blocks into a disk's bad block file, you must convert these physical addresses into logical block numbers. The /BAD=MAN and the /BAD=MAN:NOAUTO qualifiers require logical block numbers.

To convert physical addresses to logical block numbers, use the following procedure:

1. If necessary, convert the cylinder, track, and sector addresses to decimal numbers.

2. Use the following formula to define the logical block number of the bad block:

   ```
   LBN = (((cylinder number * tracks-per-cylinder)+ track number)
         * sectors-per-track) + sector number
   ```

For example, a diagnostic program has reported a data error on an RP06 disk at cylinder 198, track 5, and sector 8. An RP06 contains 19 tracks per cylinder and 22 sectors per track. Applying the formula produces the following result:

```
LB2 = (((198. * 19.) + 5.) * 22.) + 8.

    = (( 3762. + 5. ) * 22.) + 8.

    = (3767. * 22.) + 8.

    = 82874. + 8.

    = 82882.
```

The logical block number 82882. can now be used in a DSC command.

4.5 DSC MESSAGES AND ERROR RECOVERY PROCEDURES

This section defines the formats of messages displayed by the DSC programs, explains the meaning of each message, and indicates the user action needed (if any) to correct the situation that caused the message to be generated.

4.5.1 DSC Message Categories

There are four categories of DSC messages:

1. Information messages. These messages, displayed during DSC processing, provide the operator with information about the current DSC operation. For example:

   ```
   DSC -- 45 STARTING VERIFY PASS
   ```

2. Instruction messages. These messages are displayed when operator action is required to continue the current DSC operation. The operation pauses until the operator performs the requested action, then resumes. For example:

   ```
   DSC -- 44 MOUNT REEL 2 ON MTA1:  AND HIT RETURN
   ```

3. Warning messages. These messages are displayed when a condition is detected that could cause a fatal error during subsequent DSC operations or could affect the validity of the operation. DSC processing continues as warning messages are displayed. For example:

   ```
   DSC -- *WARNING* 56 OUTPUT DISK DBA1:  IS NOT BOOTABLE
   ```

4. Fatal messages. Fatal messages terminate the current DSC operation; the program prompts for another command string. You must correct the condition that caused the message and retry the DSC operation or you must terminate the DSC program. For example:

   ```
   DSC -- *FATAL* 40 I/O ERROR F ON DBA2:
   PRIVILEGE VIOLATION
   000360

   DSC2>
   ```

4.5.2 Interpreting DSC Messages

Some DSC error messages, including those classified as I/O error messages, contain error codes, the meanings of which provide supplementary information about the error condition. Table 4-2 defines these codes.

Table 4-2
Error Codes in DSC Messages

Type of Message	Code	Meaning
General Error Messages	CODE A	Failed to read storage map header
	CODE B	Input data out of phase
	CODE C	Nondata block encountered
	CODE D	Input file out of phase
	CODE E	File attributes out of phase
	CODE F	File header out of phase
I/O Error Messages	A	Reading index file bit map
	B	Reading index file header
	C	Reading storage bit map
	D	Reading boot or home block
	E	Reading file header
	F	Input (or output) device
	G	Writing index file bit map
	H	Writing storage bit map header
	I	Reading input device
	J	In input tape labels
	K	Reading file attributes
	L	Reading file header
	M	Reading index file data
	N	Reading summary data
	O	Writing file header

4.5.3 DSC Messages

The general DSC messages are listed below with explanations and suggested user actions. Section 4.5.4 lists the DSC I/O messages. In both sections the following notations are used:

FILE ID n	File identifications are displayed as two or three numbers, for example: FILE ID 000050,000002 FILE ID 000654,003456,000001
VBN n	VBNs are displayed as two numbers, for example: VBN 000000,000001
aan:	The physical device
"label"	The DSC-produced file label

1 UNDEFINED ERROR

Explanation: An unidentifiable internal error was encountered.

User Action: First, retry the operation. If the error recurs, submit a Software Performance Report (SPR).

2 CONFLICTING DEV. TYPES

Explanation: An illegal combination of device types was specified.

User Action: Check for typographical errors in device abbreviations; make sure that disks and tape drives are not specified on the same side of the command string.

3 MIXED TAPE TYPES

Explanation: Two different types of tape drive were specified in the command string.

User Action: Reenter the command specifying only the magnetic tape drive.

4 ILLEGAL SWITCH

Explanation: The command string was entered with a qualifier that cannot be used.

User Action: Reenter the command with all qualifiers correctly specified.

5 FILE LABEL TOO LONG

Explanation: A file label consisting of more than 12 characters was specified.

User Action: Correct the file label, and retry the operation.

6 SYNTAX ERROR

Explanation: An error in the command string format occurred.

User Action: Check the command, and reenter the command in the correct order.

7 DUP. DEV. NAME

Explanation: The same device was specified more than once in the command string.

User Action: Reenter the command, specifying each device only once.

8 TOO MANY DEV'S

Explanation: More than eight devices were specified on one side of the command string.

User Action: Reenter the command, specifying no more than eight devices per side.

9 DEV. aan: NOT IN SYSTEM

Explanation: The specified device is not present in the configuration of the operating system being used.

User Action: Check the device identifier that was entered in the command string, and reenter the command.

10 DEV. aan: NOT FILES-11

Explanation: The specified input device is not formatted as a Files-11 device.

User Action: Check the input device to ensure it is the one desired, and reenter the command.

11 BAD BLOCK SYNTAX ERROR

Explanation: A syntax error occurred when bad block data was entered manually.

User Action: Check the command that was entered, and reenter it correctly.

12 BAD BLOCK COUNT TOO LARGE

Explanation: Too many bad blocks were manually entered in a single group.

User Action: Check the blocks being entered. If possible, enter one large group instead of several small groups.

13 BAD BLOCK CLUSTER OUT OF RANGE

Explanation: A manually entered bad block or group of bad blocks did not exist on the output disk.

User Action: Check the numbers of the blocks entered, and reenter them correctly.

14 OUTPUT TAPE aan: NOT AT BOT

Explanation: The specified continuation tape was not at load point.

User Action: Remount or reset the tape at load point, and reenter the command.

15 OUTPUT TAPE aan: FULL

Explanation: The specified tape is full; data cannot be appended to it.

User Action: Reenter the command, and change the output tape.

16 OUTPUT TAPE aan: NOT ONLY REEL IN SET

Explanation: An illegal append operation was attempted.

User Action: Reenter the command, and either omit the Append qualifier to write to the specified tape or change tapes.

17 TAPE aan: NOT ANSI FORMAT

Explanation: If aan: is an input tape, it is not in the correct format. If an output tape, an illegal Append qualifier was specified.

User Action: For input, check the tape format and change the tape, if necessary. For output, either change tapes or omit the Append qualifier from the command string.

18 OUTPUT TAPE aan: NOT DSC TAPE

Explanation: An append operation was attempted to a tape that was not created by DSC.

User Action: Reenter the command, and either omit the Append qualifier or change tapes.

19 TAPE aan: A CONTINUATION TAPE

Explanation: If aan: is an output tape, an illegal append operation was attempted. You can use the Append qualifier only on the first volume of a tape set. If aan: is an input tape, the tape was mounted out of sequence.

User Action: Reenter the command, and change the output tape, or reenter the command, and specify the input tapes in the correct order.

20 CANNOT DETERMINE DENSITY OF TAPE aan:

Explanation: Either the tape is recorded at a density that DSC cannot use or a hardware error has occurred.

User Action: Retry the operation. If the error recurs, notify the owner of the tape that it cannot be used. If it is determined later that the tape is recorded at the correct density, contact DIGITAL Field Service to report a possible hardware error.

21 FAILED TO FIND HOME BLOCK aan:

Explanation: A read error occurred during an attempt to copy from the input disk. Either the disk is bad, the home block is bad, or the disk is not in Files-11 format.

User Action: Check the disk in question, change disk drives if possible, and reenter the command.

22 FILE STRUCTURE LEVEL ON aan: NOT SUPPORTED

Explanation: The specified DSC utility program and the structure level of the specified volume did not agree.

User Action: Replace the device, and retry the operation.

23 I/O ERROR A ON aan:

Explanation: One or more messages will accompany this message, explaining why the specified file could not be read.

User Action: Retry the operation.

24 I/O ERROR B ON aan:

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred and explaining why the file header on the device could not be read. The specified file was lost.

User Action: Retry the operation after correcting the cause of the error on the device.

25 CODE A aan:

Explanation: The file header for the storage bit map file could not be read.

User Action: The disk is unusable and therefore cannot be copied.

26 I/O ERROR C ON aan:

Explanation: One or more messages will accompany this message, explaining that an I/O error occurred during an attempt to read the specified file.

User Action: Retry the operation.

27 I/O ERROR D ON aan:

Explanation: A diagnostic message will accompany this message, indicating that a read error occurred during an attempt to read the name or boot block of the disk.

User Action: Retry the operation on a new drive.

28 RELATIVE VOLUME n OF SET NOT MOUNTED

Explanation: The specified tape is not on the system.

User Action: Mount the tape, and reenter the command.

29 Reserved

30 Reserved

31 I/O ERROR E ON aan: FILE ID n

Explanation: One or more messages will accompany this message, explaining that an I/O error occurred during an attempt to read the specified file header.

User Action: Retry the operation.

32 INPUT DEVICE aan: FILE ID n NOT PRESENT

Explanation: The specified file did not have a file header in the index file; the file was not copied.

User Action: This is a warning only. If desired, the operation can be retried on a different disk drive.

33 INPUT DEVICE aan: FILE ID n IS DELETED

Explanation: The specified file was found to be partially deleted on the input disk and was not copied.

User Action: This is a warning only. No action is required.

34 INPUT DEVICE aan: FILE ID n UNSUPPORTED STRUCTURE LEVEL

Explanation: The file's structure level recorded in the file header did not match the volume's structure level. This inconsistency is probably due to a garbled file header. There is no such file as n.

User Action: No user action is necessary.

35 INPUT DEVICE aan: FILE ID n FILE NUMBER CHECK

Explanation: An incorrect file header was read from disk causing the specified file to be lost.

User Action: Retry the operation.

36 INPUT DEVICE aan: FILE ID n FILE HEADER CHECKSUM ERROR

Explanation: Incorrect file header contents caused the specified file to be lost.

User Action: Retry the operation.

37 INPUT DEVICE aan: FILE ID n SEQUENCE NUMBER CHECK

Explanation: The sequence number was incorrect.

User Action: Retry the operation, and/or replace the disk.

38 INPUT DEVICE aan: FILE ID n SEGMENT NUMBER CHECK

Explanation: The linkage connecting file segments was broken; the specified file was lost.

User Action: Retry the operation.

39 DIRECTIVE ERROR - n

Explanation: An internal error occurred, usually the result of a system overload.

User Action: Retry the operation.

40 I/O ERROR F ON aan:

Explanation: One or more messages will accompany this message, indicating that the specified input or output device may subsequently cause an error.

User Action: This message is a warning only. No action is required unless another error message is displayed. If another error message is displayed, correct the cause of the error and reenter the command.

41 I/O ERROR I ON aan: FILE ID n, VBN n

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred that resulted in bad data being read from the specified virtual block number on the indicated device.

User Action: This is a warning message only. The block specified should be examined to determine the extent of the error.

42 VERIFICATION ERROR ON aan: FILE ID n, VBN n

Explanation: This is a warning signifying that one block of the input and output devices did not match.

User Action: When the operation is complete, you should decide whether the mismatch requires that you retry the operation.

43 BAD DATA BLOCK ON aan: FILE ID n, VBN n

Explanation: A parity error occurred during an attempt to copy the block's contents from disk. The block specified on the output disk contains erroneous data.

User Action: When the copy operation is completed, the data contained in the specified block should be examined and corrected.

44 MOUNT REEL n ON aan: AND HIT RETURN

Explanation: This is an instruction only.

User Action: Mount the volume number requested on the specified tape drive, and enter a carriage return when ready.

45 STARTING VERIFY PASS

Explanation: This is simply a message informing you that the copy operation is complete and DSC is initiating the verify pass (/VE was specified).

User Action: No user action required.

46 RESUME COPYING

Explanation: This is simply a message informing you that the verify pass is complete (/VE was specified) and DSC is continuing the copy operation.

User Action: No user action required.

47 aan: IS WRITE LOCKED. INSERT WRITE RING AND HIT RETURN

Explanation: The tape on the specified tape drive cannot be written on until a write-enable ring is inserted.

User Action: Make sure the tape is the one you want, insert the write ring, and press return.

48 INPUT FILE ON aan: WILL BE RESYNCHRONIZED

Explanation: The tape position was lost during an attempt to read the input tape. The file specified in the message, as well as some subsequent files, may be lost. Additional errors will probably occur.

User Action: Retry the operation from the beginning.

49 OUTPUT DEVICE aan: FULL FILE ID n

Explanation: The specified device is full and cannot accommodate the data following the specified file. This may mean that more data than anticipated was transferred due to an inconsistency in the input tapes. Or, the output device may contain too many bad blocks to allocate a large contiguous file.

User Action: Reenter the command, using a larger output disk.

50 **OUTPUT FILE HEADER FULL ON aan: - FILE ID n**

Explanation: Too many blocks on the output disk have caused inconsistencies in file header data. The specified file was lost.

User Action: Retry the operation with a different output disk.

51 OUTPUT FILE HEADER ON aan: NOT MAPPED - FILE ID n

Explanation: Space for the specified file header was not allocated. The file was lost.

User Action: Retry the operation; a new disk may be required.

52 I/O ERROR G ON aan:

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred during an attempt to write the specified file.

User Action: Retry the operation.

53 FAILED TO READ FILE EXTENSION HEADER ON aan: - FILE ID n

Explanation: During an attempt to copy data from the input disk, an extension header was searched for, but not found. The remainder of the specified file was lost. A problem may exist with the input disk, or a previous I/O error may have caused an inconsistency.

User Action: Retry the operation.

54 FAILED TO ALLOCATE HOME BLOCK aan:

Explanation: The home block could not be created on the specified disk device because it has too many bad blocks.

User Action: Replace the device, and reenter the command.

55 INDEX FILE ALLOCATION FAILURE aan:

Explanation: Too many bad blocks exist to allow the allocation for the specified file.

User Action: Replace the disk, and reenter the command.

56 OUTPUT DISK aan: IS NOT BOOTABLE

Explanation: Logical block number 0 of the specified disk or tape is bad.

User Action: This is a warning only. No action is required.

57 INVALID BAD BLOCK DATA aan:

Explanation: The bad block data on the output disk is invalid.

User Action: Run the BAD utility on the disk, manually enter bad block data, or reenter the command using a new disk.

58 BAD BLOCK FILE FULL aan:

Explanation: Too many bad blocks exist on the output disk.

User Action: Replace the disk, and reenter the command.

59 NO BAD BLOCK DATA FOUND aan:

Explanation: No bad block data exists for the specified output disk.

User Action: If bad block data is not desired, ignore the message. Otherwise, run the BAD program on the disk, manually enter bad block data, or reenter the command using a new disk.

60 OUTPUT DEVICE aan: IS A DIAGNOSTIC PACK. DO NOT USE IT!

Explanation: The specified output disk is a diagnostic pack and cannot be used.

User Action: Mount another output disk, and reenter the command.

61 CODE B ON aan: FILE ID n - VBN n EXPECTED, m FOUND

Explanation: The tape position was lost during an attempt to read the virtual block number specified. Some data may be lost.

User Action: Determine the extent of the error. If necessary, try the tape on another drive or create another tape.

62 CODE C ON aan: FILE ID n - VBN n

Explanation: The position of the tape was lost during an attempt to read the specified data file. Data beyond the virtual block number specified was lost.

User Action: Re-create the tape or retry the operation on a different tape drive.

63 CODE D ON aan: FILE ID n EXPECTED, m FOUND

Explanation: The tape position was lost during an attempt to read the specified tape. All of "n" and some of "m" were lost.

User Action: Retry the entire operation.

64 FAILED TO MAP OUTPUT FILE ON aan: FILE ID n, VBN n

Explanation: An inconsistency occurred during an attempt to write the specified file to the output disk. The file header did not specify the correct number of virtual blocks required to write the file and the file was lost.

User Action: Retry the operation.

65 OUTPUT DISK aan: IS TOO SMALL - n BLOCKS NEEDED

Explanation: The output disk is not large enough to accommodate the data to be transferred.

User Action: Retry the operation specifying a larger output disk.

66 I/O ERROR CODE C ON aan:

Explanation: One or more messages will accompany this message, explaining that an I/O error occurred during an attempt to read the specified file.

User Action: Retry the operation.

67 I/O ERROR CODE H ON aan:

Explanation: One or more messages will accompany this message, explaining that an I/O error occurred during an attempt to write the specified file.

User Action: Retry the operation.

68 I/O ERROR CODE J ON aan:

Explanation: One or more messages will accompany this message, explaining that an I/O error occurred during an attempt to read the tape labels on the specified device.

User Action: Retry the operation on a different tape drive.

69 INPUT TAPE ON aan: MUST BE AT BOT

Explanation: The specified tape must be at the beginning of the tape (BOT) or at its load point. This message is also displayed during a verify operation to indicate that the current volume is rewinding to enable the verify pass.

User Action: If /VE was not specified, check the tape and remount at load point.

70 WRONG INPUT TAPE ON aan: EXPECTING "label", FOUND "label"

Explanation: The input tapes were specified out of sequence.

User Action: Check the tapes and reenter them in the correct order after receiving mount instructions.

71 CODE E ON aan: AFTER FILE ID n

Explanation: This is the result of a read error from tape. During an attempt to read an attribute block, some other block was accessed. The file following the file specified in the error message was lost.

User Action: Retry the operation.

72 I/O ERROR K ON aan: AFTER FILE ID n

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred during an attempt to read the specified file.

User Action: Retry the operation.

73 I/O ERROR L ON aan: AFTER FILE ID n

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred during an attempt to read the file header.

User Action: Retry the operation.

74 INPUT TAPE aan: RESYNCHRONIZED AT FILE ID n

Explanation: The tape position was recovered. Some data preceding the file specified was lost.

User Action: This message is usually displayed with one or more error messages, all indicating that the input tape was either read incorrectly or recorded badly. The tape should be re-created and the operation reinitiated.

75 TAPE FILE "label" NOT FOUND aan:

Explanation: The input tape specified does not contain the file identified as "label."

User Action: Check the file label and the tape, and reenter the command when the correct tape and file label are specified.

76 EXPECTED EXTENSION HEADER NOT PRESENT ON aan: - FILE ID n

Explanation: A tape read error occurred, causing the specified file to be lost.

User Action: If the error message was preceded by one or more I/O warning messages, the operation should be retried. If not, the input tape is bad and should be regenerated.

77 CODE F ON aan: AFTER FILE ID n

Explanation: This is the result of a read error from tape. During an attempt to read a file header, some other block type was accessed. The file following the file specified in the error message was lost.

User Action: Retry the operation.

78 I/O ERROR M ON aan:

Explanation: One or more messages will accompany this message, explaining why the specified file could not be read.

User Action: Retry the operation.

79 INDEX FILE DATA NOT PRESENT aan:

Explanation: During an attempt to read the input tape specified, a file other than the index file was accessed due to a tape error or an I/O error.

User Action: Re-create the tape or retry the same tape on a different tape drive.

80 I/O ERROR N ON aan:

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred during an attempt to restore the index and storage map files from the specified input tape.

User Action: Retry the operation using a different input tape drive.

81 VOLUME SUMMARY DATA NOT PRESENT aan:

Explanation: Either the input tape is not a DSC tape or it contains incomplete data.

User Action: Check the tape, and reenter the command.

82 I/O ERROR O ON aan: FILE ID n

Explanation: One or more messages will accompany this message, indicating that an I/O error occurred during an attempt to write the specified file header.

User Action: Retry the operation.

83 UNSUPPORTED DSC TAPE FORMAT ON aan:

Explanation: This tape cannot be processed with this version of the DSC program.

User Action: Retry the operation. If the same failure recurs, contact DIGITAL Software Support, or submit a Software Performance Report (SPR).

4.5.4 DSC I/O Error Messages

The DSC I/O error messages are listed below.

BAD BLOCK NUMBER

Explanation: The block does not exist on the disk, an internal DSC error occurred, or the block is bad.

User Action: Retry the operation with a new disk and/or disk drive.

BAD BLOCK ON DEVICE

Explanation: A device malfunction occurred or a tape with bad data on it was used, resulting in a block containing incorrect information.

User Action: Retry the operation.

BLOCK CHECK OR CRC ERROR

Explanation: A parity error occurred indicating that bad data may have been transferred.

User Action: Retry the operation.

DATA OVERRUN

Explanation: The physical tape used was larger than expected or got out of position, or was in the wrong format.

User Action: Make sure the tape is the right one and retry the operation.

DEVICE NOT READY

Explanation: The device was not ready or not up to speed, or a blank tape was used as an input tape.

User Action: Retry the operation after checking that the device is online and correctly mounted.

DEVICE OFF-LINE

Explanation: The device is not in the system.

User Action: Check both the device and the device specification in the command string, and reenter the command.

DEVICE WRITE LOCKED

Explanation: The disk drive is write locked.

User Action: Write enable the disk drive, and reenter the command.

END OF FILE DETECTED

Explanation: The tape position was lost.

User Action: Retry the operation.

END OF TAPE DETECTED

Explanation: The tape position was lost.

User Action: Retry the operation.

END OF VOLUME DETECTED

Explanation: The tape position was lost.

User Action: Retry the operation.

FATAL HARDWARE ERROR

Explanation: A hardware malfunction occurred.

User Action: Retry the operation; if the error recurs call DIGITAL Field Service.

ILLEGAL FUNCTION

Explanation: An operation was attempted, but DSC cannot determine what it was.

User Action: Retry the operation. If the same failure recurs, contact DIGITAL Software Support or submit a Software Performance Report (SPR).

INSUFFICIENT POOL SPACE

Explanation: The operating system is overloaded.

User Action: Retry the operation.

PARITY ERROR ON DEVICE

Explanation: A device malfunction or media incompatibility occurred.

User Action: Retry the operation.

PRIVILEGE VIOLATION

Explanation: A device has been mounted as Files-11.

User Action: Dismount the disk, mount it as a foreign volume, and retry the operation.

UNKNOWN SYSTEM ERROR

Explanation: An undefinable I/O error occurred.

User Action: Retry the operation.

CHAPTER 5

BAD BLOCK LOCATOR UTILITY

The Bad Block Locator utility (BAD) determines and records the logical block numbers and location of faulty blocks that cannot reliably store data. BAD can be used on the following block-structured volumes:

- TU58 DECtape II data cartridge
- RK07 disk cartridges
- RL02 disk cartridge
- RM03 disk packs
- RP06 disk packs
- RX01/02 floppy diskettes

Usually, BAD tests block-structured volumes that have not been initialized. After BAD locates and records the bad blocks, you issue the DIGITAL Command Language (DCL) command INITIALIZE so that the operating system will allocate the faulty blocks to a special file. In this way, users are protected from accessing these faulty blocks for their files.

Section 5.1 below explains how BAD locates and records bad blocks; Section 5.2 explains how the INITIALIZE command allocates bad blocks. The remaining sections of this chapter describe now to invoke BAD, the BAD command line format and qualifiers, and the messages BAD can issue.

5.1 LOCATING AND RECORDING BAD BLOCKS

BAD locates bad blocks on a volume by testing whether the same data that is written into blocks can be read out. When it finds a bad block, BAD writes the address of that block into the bad block descriptor (described in Section 5.1.2).

5.1.1 Locating Bad Blocks

To test the blocks on a volume, BAD:

- Writes a test pattern onto each block
- Reads the contents of blocks into a buffer
- Compares the data in the buffer with the data it wrote into the blocks

If the data does not compare exactly, one or more blocks in the group of blocks are bad and cannot reliably store data. In this case, BAD will repeat the reading, writing, and comparing operations on each block in the group to determine the bad block(s).

5.1.2 Recording Bad Blocks

When BAD locates a bad block, it records the address of the block. Consecutive bad blocks are recorded as single entries. After it finishes testing the disk, BAD writes the addresses of the bad blocks into an area called the bad block descriptor.

5.1.2.1 **Location of the Bad Block Descriptor** - The location of the bad block descriptor depends on whether the volume is a last-track device. Last-track devices store bad block data on the last track of the disk.

The first half of the track is reserved for the Manufacturer's Detected Bad Sector File (MDBSF). The MDBSF stores the bad blocks discovered by the manufacturer when the device was originally formatted.

The second half of the track is reserved for the Software Detected Bad Sector File (SDBSF). The bad block descriptor is located here.

Last-track devices are:

- RK07, RL02 disk cartridges
- RM03/05 disk cartridges

Other devices (non-last-track devices) do not set aside the last track of the disk to store bad block information. Instead, BAD creates the bad block descriptor on the last good block of the disk. There must be at least one reliable block in the last 256 blocks of the volume for BAD to generate the bad block descriptor.

Non-last-track devices are:

- RP06 disk packs
- RX01/02 floppy diskettes
- TU58 DECtape II data cartridges

5.1.2.2 **Format of the Bad Block Descriptor** - If the volume is a last-track device, each bad block descriptor entry contains the cylinder, track, and sector addresses of the faulty block. The bad block descriptor can record a maximum of 126 entries.

On volumes that are not last-track, bad block descriptor entries contain the number of bad blocks minus 1 and bits 0 through 23 of the logical block number (LBN) of the faulty block or sequence of faulty blocks. A single entry can address one bad block or several contiguous bad blocks. The bad block descriptor on non-last-track devices can contain a maximum of 102 entries.

For both last-track and non-last-track devices, once the maximum number of entries is exceeded, BAD terminates with an error message.

5.2 ALLOCATING BAD BLOCKS

After you run BAD, the final step in processing bad block data is to issue the DCL command INITIALIZE. INITIALIZE changes the volume from unstructured format to Files-11 format and allocates the bad blocks found by BAD to a special file on the volume called [0,0]BADBLK.SYS. Once they are allocated to BADBLK.SYS, the faulty blocks cannot be used by other files. For further information on Files-11 format and the INITIALIZE command, see the VAX/VMS Command Language User's Guide.

5.3 INVOKING AND TERMINATING BAD

When running BAD to test a device, keep in mind that:

- The device cannot be accessed by other programs
- The device cannot be mounted as a Files-11 volume
- The device is always purged by BAD's testing procedure; any information stored on the disk is destroyed

To ensure that the device is not accessed by any other programs, you must allocate the device with the DCL command ALLOCATE. See the VAX/VMS Command Language User's Guide for more information on the ALLOCATE command.

After you have allocated the device, you must give the DCL command MOUNT with the /FOREIGN qualifier. When the device is mounted foreign, the operating system does not recognize it as a Files-11 volume and BAD can execute.

There is no way to test the volume for bad blocks without destroying its contents. However, you can update the bad block descriptor without wiping out the volume by using the BAD qualifier /UPDATE. This qualifier is described in detail in Section 5.5.5.

To invoke BAD, enter the following command in response to the DCL prompt:

```
$ RUN SYS$SYSTEM:BAD
```

The utility responds with the prompt:

```
BAD>
```

You can now enter any BAD command string (Section 5.4). To return to DCL at any time, type <CTRL/Z>.

You can also invoke BAD by using the RSX-11M Monitor Console Routine (MCR) command:

```
$  MCR BAD [device-name]
```

BAD issues the prompt BAD>. The device name format is the same as described in Section 5.4.

5.4 BAD COMMAND STRING

The BAD command string has the following format:

```
BAD> device-name:[/qualifier...]
```

device-name

The device containing the volume on which BAD will be run. The device name has the form:

```
devu
```

where

```
dev = 2- character alphabetic device code

  u = 1- or 2-digit octal device unit number
```

The colon (:) acts as the device-name terminator and must follow the device name. BAD does not recognize alphabetic controller designators. You must convert them to RSX-11M unit numbers when specifying devices. For information on conversion between VAX/VMS native mode unit numbers and compatibility mode unit numbers, see the explanation of mapping physical device names in the VAX-11/RSX-11M User's Guide.

/qualifier(s)

The BAD qualifiers that modify BAD operation. Multiple qualifiers are entered on the same command line; no separators are required. Section 5.5 discusses the BAD qualifiers in detail.

5.4.1 Running BAD Interactively from Your Terminal

The example below shows the sequence of commands that you should use when running BAD interactively from your terminal:

```
$ ALLOCATE DBA2:
_DBA2: ALLOCATED
$ MOUNT/FOREIGN DBA2:
%MOUNT-I-MOUNTED      mounted on _DBA2:
$ RUN SYS$SYSTEM:BAD
BAD>DB2:
BAD -- TOTAL NO. OF BAD BLOCKS = 2.
BAD> CTRL/Z
$
```

The ALLOCATE command requests the allocation of a specific disk drive, DBA2. The response from the ALLOCATE command indicates that the device was successfully allocated. The MOUNT/FOREIGN command mounts the disk volume as a foreign disk. The MOUNT command response indicates that DBA2 was successfully mounted. The RUN SYS$SYSTEM:BAD command invokes BAD. Specifying DB2 causes BAD to analyze each block on the disk volume and record the bad blocks. After BAD has tested all the blocks, it indicates that the number of bad blocks on DBA2 is 2. You exit from BAD by entering <CTRL/Z> in response to the BAD> prompt.

5.4.2 Running BAD from Command Procedures

You can invoke the BAD utility from a VAX/VMS command procedure. The following is a command procedure, named STEPS.COM, that invokes BAD and gives other DCL commands.

```
$ ALLOCATE DBB1:
$ MOUNT/FOREIGN DBB1:
$ RUN SYS$SYSTEM:BAD
DB21:/LI
$ DISMOUNT/NOUNLOAD
$ INITIALIZE DBB1:
```

To call the command procedure, type the following in response to the DCL prompt:

```
$ @STEPS
```

The operating system executes the commands in the order they are given within the command procedure.

Note that because you are calling the command procedure from DCL, the default file type is COM and need not be specified. For a thorough discussion of command procedures, refer to the VAX/VMS Guide to Using Command Procedures.

BAD also allows you to use a command procedure execute a series of BAD commands. The following example is a command procedure, named BADCMD.CMD, that contains the commands BAD is to execute. These commands are explained in Section 5.5.2.

```
DM2:/MAN
45
102
CTRL/Z
```

To call the command procedure, type:

```
$ MCR BAD

BAD> @BADCMDS
```

The default file type in this example is CMD because you are calling the command procedure from the MCR command interpreter.

BAD is invoked, performs the requested functions, and exits. Note that you can omit the file type when you call the command procedure. You can call up to three other command procedures from within one command procedure.

5.5 BAD QUALIFIERS

BAD provides five qualifiers which, when added to the command string, modify BAD operation. Table 5-1 lists the BAD qualifiers and gives a summary their functions.

Table 5-1
BAD Qualifiers

Qualifier	Notation	Function
List	/LI	Lists logical block numbers of bad blocks at your terminal
Manual	/MAN	Enters specific bad blocks to the bad block descriptor and tests the disk volume
Override	/OVR	Converts last-track devices to non-last-track devices
Retry	/RETRY	Enables the device driver to correct soft errors
Update	/UPD	Updates the bad block descriptor without testing the disk

The following sections describe each of these qualifiers in detail.

5.5.1 The List Qualifier

The List qualifier (/LI) causes all bad blocks to be listed by logical block number (LBN) on your terminal. Each time BAD encounters a faulty block, it writes the following message:

```
BAD -- BAD BLOCK FOUND - LBN= n
```

The value of n is the logical block number (LBN) of the block in decimal.

This qualifier is valid for all devices. If you do not specify the LIST qualifier, BAD will execute without listing the bad blocks at your terminal.

Example

```
BAD> DB2:/LI

BAD -- DB2:  BAD BLOCK FOUND - LBN= 20663
```

5.5.2 The Manual Qualifier

The Manual qualifier (/MAN) permits you to enter specific blocks to the bad block descriptor of an unformatted volume. You may want to use this qualifier to allocate specific blocks or a series of blocks so that they will not be used by other files.

When you use the Manual qualifier, BAD prompts for the logical block number of the blocks you want to enter:

```
BAD> LBN(S)=
```

You can specify a single block or consecutive blocks in the form:

```
lbn:[count]
```

The value of lbn is the logical block number in decimal and count is the number of consecutive blocks beginning at the specified LBN. You must use the colon (:) when specifying consecutive blocks and separate different LBNs or consecutive LBNs on a single line by a space, comma, or tab. Both lbn and count default to decimal unless they are preceded by the pound sign (#) to indicate octal.

After you finish entering specific bad blocks, type <CTRL/Z> or <ESC>. BAD enters the blocks you have specified into the bad block descriptor and then tests the volume for bad blocks.

If you do not specify an LBN in response to the prompt but instead press <RETURN>, BAD lists the contents of the bad block descriptor in the format:

```
lbn:count
```

The value of lbn is the initial LBN of a possible sequence of bad blocks and count is the number of bad blocks in the sequence (in decimal). Single blocks are represented in the same format as a sequence of bad blocks.

Examples

1.
```
BAD>DB0:/MAN
BAD> LBN(S)= 45 CTRL/Z
BAD -- DB0:  TOTAL BAD BLOCKS= 2.
```

BAD enters the block represented by LBN 45 into the bad block descriptor, tests the disk for bad blocks, and reports the total number of bad blocks that it found at your terminal. (BAD does not include manually entered blocks in this total).

2.
```
BAD>DM2:/MAN
BAD> LBN(S)= 100:2,3 200:10. ESC
BAD -- DM2:  TOTAL BAD BLOCKS= 13.
```

BAD enters blocks 100, 101, 3, and 200 through 209 into the bad block descriptor.

3.
```
BAD>DM3:/MAN
BAD> LBN(S)= RET
000100:002
000003:001
000200:100
```

BAD lists all blocks in the bad block descriptor by logical block number and count.

5.5.3 The Override Qualifier

The Override qualifier (/OVR) causes BAD to ignore last-track information (the MDBSF and SDBSF, described in Section 5.1.2.1).

When specified, /OVR creates a bad block descriptor on the last good block before the last track of the disk, but does not generate a message at your terminal. If the last track does not contain a bad block descriptor, or if you suspect that the last track is faulty, use /OVR.

The Override qualifier converts last-track devices to non-last-track devices; thus, it is valid only on last-track devices.

5.5.4 The Retry Qualifier

The Retry qualifier (/RETRY) enables the device driver to correct soft errors. A soft error is a type of hardware error that causes good blocks to appear faulty. If /RETRY is not specified, BAD will prevent the device driver from correcting soft errors, and blocks mistakenly identified as bad will not be discovered.

There is no example for this qualifier because /RETRY produces no output when it is enabled.

5.5.5 The Update Qualifier

The Update qualifier (/UPD) enters additional blocks to the bad block descriptor without testing the volume. Use the Update qualifier when you want to update the bad block descriptor without destroying the contents of the volume.

The Update qualifier prompts for additional bad blocks in the same way as the Manual qualifier. When you have finished specifying the logical block numbers of the blocks you want entered to the bad block descriptor, type <CTRL/Z> or <ESC>. BAD will update the descriptor and exit.

If you do not specify an LBN in response to the prompt, but instead press <RETURN>, BAD lists the contents of the bad block descriptor in the format

```
lbn:  count
```

as described in Section 5.5.2.

Example

```
$ RUN SYS$SYSTEM:BAD
BAD> DM1:/UPD
BAD> LBN= 123
BAD> CTRL/Z
```

The Update qualifier causes BAD to enter logical block number 123 into the bad block descriptor. Entering <CTRL/Z> returns control to DCL without testing the device.

5.6 BAD MESSAGES

This section describes the diagnostic messages generated by BAD as it executes. Each message begins with:

BAD -- devu:

where devu: is the device name of the block-structured volume that BAD is testing.

BAD BLOCK FILE NOT FOUND

Explanation: The bad block descriptor cannot be read.

User Action: This message occurs when you have specified the Update qualifier. It means that you cannot use /UPD on the volume without re-initializing the device. (When the volume is initialized, all previous data stored on the device is destroyed; see the VAX/VMS Command Language User's Guide for information on the INITIALIZE command.)

BAD BLOCK FILE OVERFLOW

Explanation: BAD detected more than the maximum number of bad blocks (126 for last-track devices and 102 for non-last-track devices). This message usually indicates a device unit failure.

User Action: Either the volume is bad or the drive requires maintenance; contact your DIGITAL Field Service Representative.

BAD BLOCK FOUND - LBN= lbn

Explanation: Bad blocks are reported in this format when you specify the List qualifier; lbn is the logical block number in decimal.

User Action: None. This is an informational message and applies only to the List qualifier.

BLOCK 0 BAD - DO NOT USE AS SYSTEM DISK

Explanation: This is a warning message that can be ignored on VAX/VMS systems. Unlike RSX-11M and IAS systems, VAX/VMS does not use block zero for bootstrapping purposes.

User Action: Ignore the message.

COMMAND I/O ERROR

Explanation: The operating system detected a hardware transmission error from the keyboard.

User Action: Retype the command.

COMMAND TOO LONG

Explanation: The command line you typed is longer than 80 characters.

User Action: Shorten the command line.

DEVICE IS AN ALIGNMENT CARTRIDGE

Explanation: The factory-written label on the last track of a last-track device indicates that the device is an alignment cartridge.

User Action: Mount and process another device.

DEVICE NOT IN SYSTEM

Explanation: The requested device was not made part of the system generation or the device unit does not exist on the host configuration.

User Action: Reconfigure the system with the SYSGEN utility (see the VAX/VMS System Manager's Guide for details on SYSGEN).

DEVICE NOT READY

Explanation: BAD cannot access the device because the unit has not reached operating speed.

User Action: Allow the unit to reach operating speed and reenter the command line.

DUPLICATE BLOCK NUMBER - lbn

Description: The logical block number you entered is already present in the bad block descriptor.

User Action: Enter another block number. This message applies only to the Manual and Update qualifiers.

FAILED TO ATTACH

Explanation: BAD cannot gain control of the unit to be tested.

User Action: The device is allocated to another user. Mount the disk on another device unit.

FAILED TO READ MANUFACTURER'S BAD SECTOR FILE

Explanation: A disk-read hardware error has prevented BAD from reading the MDBSF of a last-track device.

User Action: Run BAD again and specify the Override qualifier.

FAILED TO READ SOFTWARE BAD SECTOR FILE

Explanation: BAD cannot read the Software Detected Bad Sector File with the Update qualifier enabled.

User Action: Run BAD again and specify the Override qualifier.

FAILED TO WRITE BAD BLOCK FILE

Explanation: BAD cannot make entries into the bad block descriptor. This condition is usually caused by a disk write error.

User Action: Run BAD again. If the problem persists, the volume should be discarded.

FATAL HARDWARE ERROR

Explanation: A machine hardware problem is preventing BAD from running.

User Action: Contact your DIGITAL Field Service Representative.

HANDLER/DRIVER MISSING

Explanation: The device driver associated with the device unit that you specified is not loaded in the operating system.

User Action: Load the device driver with the SYSGEN utility (described in the VAX/VMS System Manager's Guide). For further information on device drivers, consult the VAX/VMS Guide to Writing a Device Driver.

ILLEGAL DEVICE

Explanation: BAD does not run on volumes that are not block structured (for example, magnetic tapes).

User Action: Mount and run BAD on a block-structured volume.

INVALID BLOCK NUMBER - n

Explanation: You entered an invalid logical block number.

User Action: Type another value and reenter the command lines. This message applies only to the Manual and Update qualifiers.

INVALID SWITCH

Explanation: BAD does not recognize the qualifier you specified.

User Action: Enter a valid BAD qualifier (see Section 5.5).

MANUFACTURER'S BAD SECTOR FILE CORRUPT

Explanation: The MDBSF is improperly formatted. This message applies only to last-track devices.

User Action: Contact your DIGITAL Field Service Representative.

PRIVILEGE VIOLATION

Explanation: BAD accessed a device that was already mounted by another user.

User Action: Mount the volume you want BAD to test on another device unit.

SYNTAX ERROR

Explanation: BAD detected a syntax error on the command line.

User Action: Determine the correct syntax and reenter the command line.

TOTAL NO. OF BAD BLOCKS = n

Explanation: This message indicates the total number of bad blocks on the volume. This message appears when BAD finishes testing the volume.

User Action: Write the bad block count on the device label.

UNRECOVERABLE ERROR n

Explanation: BAD is terminated by an I/O error. The value of n represents the I/O error number returned by the device driver.

User Action: Contact your DIGITAL Field Service Representative.

WRITE-LOCKED

Explanation: BAD attempted to run on the write-locked disk volume.

User Action: Mount the device without the Nowrite qualifier to unlock the device. See the VAX/VMS Command Language User's Guide for information on the MOUNT command qualifiers.

CHAPTER 6

FILE STRUCTURE VERIFICATION UTILITY

The File Structure Verification Utilities (VFY1 and VFY2) check the readability and validity of Files-11 Structure Level 1 and 2 volumes. With the addition of a qualifier, VFY can also:

- Print out the number of available blocks on a Files-11 volume
- Search for "lost files"
- List all files in the index file
- Mark as "used" all blocks that appear available but actually are allocated to a file
- Rebuild the storage bit map to reflect information in the index file
- Restore files marked for deletion
- Perform a read check on every allocated block on the volume

The sections that follow describe validity checking and error recovery, how to invoke VFY, the VFY command line format and qualifiers, and VFY error messages.

6.1 VALIDITY CHECKING

If you do not specify a qualifier on the VFY command line, VFY performs a validity check of the volume mounted on a specified device unit. VFY first checks the integrity of all the file headers contained in the index file of the volume.

Each file on the volume has a file header that describes properties of the file and its physical location on the volume. The file headers are part of the index file [0,0]INDEXF.SYS, which is created when the volume is initialized. VFY makes sure that blocks referenced in the map area of each file header are reported as allocated in the storage bit map file [0,0]BITMAP.SYS.

As it verifies the volume, VFY reports any file errors either at your terminal or in a specified listing file (see Section 6.4). VFY identifies different file errors. Each message is preceded by a file identification line that identifies the file containing the error:

FILE ID n file-spec OWNER [uic]

n	The file identification number assigned to the file by VAX/VMS when it creates the file
file-spec	The name, type and version number of the file that contains the error
uic	The user identification code of the owner of the file

One or more of the messages listed below follows the file identification line.

I/O ERROR READING FILE HEADER-ERROR CODE n

VFY failed to read the file header for the specified file identification. One of the following conditions exists:

- the device is not mounted
- the device is offline
- the hardware has failed
- the header block is bad

BAD FILE HEADER

Software checks on the validity of the file header indicate that the header has been corrupted. The file is permanently damaged.

MULTIPLE ALLOCATION n,m

The specified logical block number is allocated to more than one file. VFY indicates the logical block number (LBN) as two octal integers n and m representing the low- and high- order bits of the LBN.

If VFY detects a multiply-allocated block, it finishes the validity check and scans the volume again to identify which files share each block. After it lists each multiply-allocated block, VFY prints a summary line for the file as follows:

SUMMARY:MULT=n, FREE=n, BAD=n.

MULT	The number of multiple block allocations
FREE	The number of blocks marked free that should have been allocated
BAD	The number of bad retrieval pointers in the file header

For information about deleting multiply-allocated blocks, see Section 6.2.2.

BLOCK IS MARKED FREE n,m

The specified LBN is allocated to the indicated file but is not marked as allocated in the storage bit map (see Section 6.2.3).

BAD BLOCK NUMBER n,m

The specified block number was found in the header for this file but is illegal for the device (out of range). This indicates a corrupted file header.

FILE IS MARKED FOR DELETE

The operating system failed while the specified file was being deleted. The deletion was not completed and the file header still exists (see Section 6.2.1).

HEADER MAP OUT OF SYNC

VFY detected an error in the header map area which also indicates a corrupted file header.

You can suppress VFY output on a terminal device by typing <CTRL/O>.

6.2 FILE ERROR RECOVERY

You can use the file error information obtained through the validity check to correct file errors on the volume. The following sections discuss how to delete and restore files marked for deletion, how to eliminate free and multiply-allocated blocks, and how to recover lost blocks.

6.2.1 Restoring Files Marked for Deletion

If VAX/VMS fails before it finishes deleting a file, you can use VFY to restore the file or resume the deletion process.

To restore a file marked for deletion, run VFY specifying the Delete qualifier (Section 6.5.1) to reset the marked-for-deletion indicators in the file headers. Once the deletion indicators has been reset, run VFY specifying the Lost qualifier (Section 6.5.4) to scan the entire file structure. You may not be able to restore the entire file because the operating system may have deleted part of the file before it failed.

Once you obtain the file identification of a marked-for-deletion file, you can finish the deletion process by running the Peripheral Interchange Program (PIP). Because PIP is an RSX-11M utility, you must use the VAX/VMS MCR interface to invoke it. Enter the following command at the DCL prompt:

```
$ MCR PIP
```

THE PIP utility responds with the prompt:

```
PIP>
```

Specify the File identification you obtained from the VFY output (see Section 6.1) in response to the PIP> prompt:

```
PIP>/FI:12:20/DE
PIP -- FAILED TO MARK FILE FOR DELETE-NO SUCH FILE
```

In the above example, the file with file identification 12,20 is deleted from the default device. The PIP error message appears because the file system denies the existence of files already marked for deletion. However, the file is deleted.

If you have restored or deleted files, you should update the volume's storage bit map by running VFY with the Rebuild qualifier (see Section 6.5.6).

6.2.2 Deleting Multiply-Allocated Blocks

VFY reports all files that contain multiply-allocated blocks (see Section 6.1). Once you have the file specification of these files, you can eliminate them with PIP, as described in Section 6.2.1, until there are no more files that share blocks.

Be careful when deleting multiply-allocated files. After you have deleted the files, run VFY again to ensure that all of the files with multiply-allocated blocks have been eliminated.

6.2.3 Eliminating Free Blocks

After you have purged the files of multiply-allocated blocks, eliminate blocks that are erroneously marked as free in the storage bit map. To correct the storage bit map, run VFY again and specify the Update qualifier (Section 6.5.7). This qualifier allocates all blocks that should have been marked as allocated.

Once you have cleared the volume of multiply-allocated blocks and blocks mistakenly marked free, it is safe to create new files and extend existing files. However, if multiply-allocated and "free" blocks still exist, the volume may contain files whose blocks are overwritten by multiple allocation.

6.2.4 Recovering Lost Blocks

To determine whether any blocks on a file-structured volume have been lost, examine the last two lines of output from the validity check. The last two lines of output give the free space on the volume. The first of these two lines tells how much room is available according to the index file (the number of blocks not in use). The second line specifies how much room is available according to the storage bit map. Assuming there are no other errors, these two figures should agree.

If the index file indicates that more blocks are free than the storage bit map, those blocks are "lost" in the sense that they appear to be allocated, but no file contains them. Run VFY again and specify the Rebuild qualifier (Section 6.5.6) to recover these lost blocks.

6.3 INVOKING VFY

When running VFY to validate a volume's structure, keep in mind that:

- No other activity should occur on the volume while VFY is executing. In particular, activities that create new files, extend existing files, or delete files should not be attempted while VFY is executing a function.

- Do not abort VFY if you have specified the Delete, Rebuild, or Update qualifier. These qualifiers modify the index file and the storage bit map; if you prevent them from completing their functions, you may seriously endanger the integrity of the volume.

- Before you run VFY, the volume must be mounted as a Files-11 structured volume. The volume can be write-locked if it is not the system volume or if the required scratch file is directed to another file-structured volume and you are just running a consistency check.

To invoke VFY, enter one of the following command lines, as appropriate:

Files-11 Structure Level 1 Format

```
$ RUN SYS$SYSTEM:VFY1
```

Files-11 Structure Level 2 Format

```
$ RUN SYS$SYSTEM:VFY2
```

The utility responds with the prompt:

```
VFY>
```

You can now enter any VFY command string (Section 6.4). To return to DCL at any time, type <CTRL/Z>.

You can also invoke VFY by using the RSX-11M Monitor Console Routine (MCR) command. Enter one of the following command lines, as appropriate:

```
$ MCR VFY1

$ MCR VFY2
```

Using the MCR command, you can enter a VFY command string on the initial command line that invokes VFY. For example:

```
$ MCR VFY1 command-string
```

VFY will execute this command string and return control to DCL.

6.4 VFY COMMAND STRING

The VFY command string has the following format:

```
VFY> [list-file-spec, scratch-device-name:=] [input-device-name:] [/qualifier]
```

list-file-spec

The listing file to which VFY output will be directed. If you omit this parameter, VFY output is displayed at your terminal.

scratch-device-name

The device on which the scratch file produced by VFY is to be written. The scratch device name has the format:

```
devcu
```

where

dev	2-character alphabetic device code
c	1-character alphabetic controller designator
u	1- or 2-digit device unit number

When VFY validity checks or scans for lost files, it creates a scratch file. This file is not entered in any directory, and thus is transparent to the user. VFY automatically deletes the scratch file when it finishes processing the volume.

If you omit this parameter, the system disk (SYS$DISK) is used automatically.

If you suspect that the default disk is faulty, use this parameter to force the scratch file to another device. VFY does not create a scratch file if you run it with either the Free or List qualifiers.

input-device-name

The volume to be verified. This parameter has the same format as the scratch device name.

/qualifier

One of the VFY qualifiers that specify the function to be performed. Only one qualifier can be specified per command string. If you specify more than one, an error occurs. If no qualifier is specified, VFY validates the structure of the volume mounted on the specified device. See Section 6.5 for a complete description of each qualifier.

6.5 VFY QUALIFIERS

Table 6-1 summarizes the VFY qualifiers. The following sections describe them in greater detail.

Table 6-1
VFY Qualifiers

Qualifier	Notation	Function
Delete	/DE	Resets marked-for-deletion indicators
Free	/FR	Indicates the number of available blocks on the volume, the number of used blocks on the volume, and the total number of blocks on the volume
List	/LI	Lists the entire index file by file identification
Lost	/LO	Scans the entire file structure for files that are not in any directory
Read Check	/RC[:n]	Checks readability of every block of every file on the entire volume
Rebuild	/RE	Recovers blocks that appear to be allocated but are not contained in a file
Update	/UPD	Allocates blocks that appear to be available but have been allocated to a file

6.5.1 **The Delete Qualifier**

The Delete qualifier (/DE) resets the marked-for-delete indicator in the file header of a file that was marked for deletion, but never actually deleted.

The volume being deleted must be write-enabled; VFY requires write access to the index file [0,0]INDEXF.SYS.

VFY must be running under a system user identification code (UIC).

Do not abort VFY if you have specified the Delete qualifier.

6.5.2 **The Free Qualifier**

The Free qualifier (/FR) displays on your terminal a message indicating the available space on a specified volume. The message has the form:

```
devcu:  HAS n.  BLOCKS FREE, n.  BLOCKS USED OUT OF n.
```

6.5.3 **The List Qualifier**

The List qualifier (/LI) lists the entire index file by file identification. The output for each file specifies the file number, file sequence number, file name, and owner. A typical index file listing is illustrated in Figure 6-1.

```
VFY>DK:/LI
LISTING OF INDEX ON DK0:

FILE ID 000001,000001 INDEXF.SYS;1    OWNER [1,1]
FILE ID 000002,000002 BITMAP.SYS;1    OWNER [1,1]
FILE ID 000003,000003 BADBLK.SYS;1    OWNER [1,1]
FILE ID 000004,000004 000000.DIR;1    OWNER [1,1]
FILE ID 000005,000005 CORIMG.SYS;1    OWNER [1,1]
FILE ID 000006,000006 001001.DIR;1    OWNER [1,1]
FILE ID 000007,000007 001002.DIR;1    OWNER [1,2]
FILE ID 000010,000010 EXEMC.MLB;1     OWNER [1,1]
FILE ID 000011,000011 RSXMAC.SML;1    OWNER [1,1]
FILE ID 000012,000012 NODES.TBL;1     OWNER [1,1]
FILE ID 000013,000036 QIOSYM.MSG;311 OWNER [1,2]
FILE ID 000014,000037 F4PCOM.MSG;1    OWNER [1,2]
```

Figure 6-1 VFY Index File Listing

6.5.4 The Lost Qualifier

The Lost qualifier (/LO) scans the entire file structure looking for files that are not in any directory and, are lost in the sense that they cannot be referenced by file name. VFY creates a list of the files and enters them in the lost file directory [1,3].

Before you use the /LO qualifier, see if the volume has the directory [1,3]. If this directory does not exist on the volume, create it using the DCL command CREATE/DIRECTORY, explained in the VAX/VMS Command Language User's Guide.

6.5.5 The Read Check Qualifier

The Read Check qualifier (/RC[:n]) checks to ensure that every block of every file on a specified volume can be read.

The optional parameter [:n] is the blocking factor which indicates the number of file blocks to be read at a time. The default value is the maximum number of blocks in memory that are available to VFY.

Because Read Check is a read-only operation, the volume can be write-locked.

When VFY detects an error, it identifies the file in the following manner:

FILE ID n file-spec. blocks used/blocks allocated

VFY prints first the file identification line, then an error message. If a blocking factor other than 1 is in use, VFY issues the following message:

ERROR STARTING AT VBN n LBN n - ERROR CODE -err

VBN n is the virtual block number that marks the start of the error, LBN n is the logical block number, and err is a negative number that represents an error code.

After VFY prints the first error message, it prints out one or more error messages indicating the exact block or blocks in error. These error message lines appears in the following format:

```
ERROR AT VBN n - ERROR CODE -err
```

If the VBN of the unreadable block listed in the ERROR AT line is beyond the block-used-count, the data portion of the file is all right.

The negative number -err is -4 to indicate a device parity error. Other error codes are contained in the VAX/VMS System Messages and Recovery Procedures Manual.

If VFY does not display an ERROR AT line, it has failed to read multiple blocks, but individual blocks are still readable.

6.5.6 The Rebuild Qualifier

The Rebuild qualifier (/RE) recovers lost blocks, that is, blocks that appear to be allocated but which are not contained in any file.

Before you can specify Rebuild, you must remove all multiply-allocated blocks from the volume.

The volume being updated must be write-enabled; VFY requires write-access to the storage bit map [0,0]BITMAP.SYS.

You must be running under a system UIC, or be the owner of the volume.

Do not abort VFY if you have specified the Rebuild qualifier.

6.5.7 The Update Qualifier

The Update qualifier (/UPD) allocates all blocks that appear to be available but are actually allocated to a file.

Files with multiply-allocated blocks must be deleted from the file structure before the update can be run.

The volume being updated must be write-enabled; VFY requires write-access to the storage map [0,0]BITMAP.SYS.

VFY must be running under a system user identification code UIC.

The scratch file should be on another volume. If this is impossible, the volume must be dismounted immediately after VFY terminates. The procedure you should follow is the same for Update as it is for Rebuild. VFY issues a detailed message specifying the scratch file to be deleted.

Do not abort VFY if you have specified the Update qualifier.

6.6 VFY MESSAGES

This section describes in detail the messages generated by VFY when it encounters various kinds of errors. Messages are issued in the format:

VFY -- message

COMMAND SYNTAX ERROR

Explanation: The command entered did not conform to command syntax rules.

User Action: Retype the command line using the correct syntax.

FAILED TO ALLOCATE SPACE FOR TEMP FILE

Explanation: The volume specified for the scratch file is full.

User Action: Delete all unnecessary files on the volume and rerun VFY.

FAILED TO ATTACH DEVICE

or

FAILED TO DETACH DEVICE

or

ILLEGAL DEVICE

Explanation: The file specification entered contains an invalid device.

User Action: Retype the command line with the correct device specified.

FAILED TO CLOSE DIRECTORY FILE

See I/O ERROR messages.

FAILED TO ENTER FILE

Explanation: One of the following conditions may exist:

- VFY is not running under a system UIC.
- The device is not online.
- The device is not mounted.
- The hardware has failed.

User Action: Determine which of the above conditions caused the message and correct that condition, then reenter the command line.

FAILED TO OPEN DIRECTORY FILE

See OPEN FAILURE message.

ILLEGAL SWITCH

Explanation: The qualifier you specified is not a valid VFY qualifier or you used a valid qualifier incorrectly.

User Action: Reenter the command line and specify the correct qualifier.

I/O ERROR ON INPUT FILE -- HANDLER ERROR CODE n

or

I/O ERROR ON OUTPUT FILE -- HANDLER ERROR CODE n

or

I/O ERROR READING DIRECTORY FILE -- HANDLER ERROR CODE n

or

FAILED TO CLOSE DIRECTORY FILE -- HANDLER ERROR CODE n

Explanation: In these messages, n is the handler error code number. For an explanation of handler error codes, see the VAX/VMS System Messages and Recovery Procedures Manual.

One of the following conditions may exist:

- The device is not online.
- The device is not mounted.
- The hardware has failed.

User Action: Determine which of the above conditions caused the message and correct that condition, then reenter the command line.

OPEN FAILURE ON BIT MAP

or

OPEN FAILURE ON INDEX FILE

or

OPEN FAILURE ON LISTING FILE

or

OPEN FAILURE ON TEMPORARY FILE

or

FAILED TO OPEN DIRECTORY FILE

Explanation: One of the following conditions may exist:

- VFY is not running under a system UIC.
- The named file does not exist in the specified directory.
- The volume is not mounted.
- The specified directory does not exist.

User Action: Determine which of the above conditions caused the message and correct that condition, then retype the command line.

THEY ARE STILL LOST, COULD NOT FIND DIRECTORY

Explanation: The lost file directory [1,3] is not present on the volume.

User Action: Use the MCR UFD command to enter UFD [1,3] on the volume. (See the VAX-11/RSX-11M User's Guide for details on MCR commands.)

CHAPTER 7

LIBRARIAN UTILITY

The Librarian is a utility that allows you easy access to libraries. Libraries are indexed files that contain frequently used modules of code or text. There are four different types of libraries -- object, macro, help, and text. The library type indicates the type of module that the library contains. Section 7.1 describes the four types of libraries. Each library also contains indexes that store information about the library's contents, including the type, location, and modification history of the individual modules. Section 7.1.2 describes library indexes.

The Librarian consists of two parts: (1) the DIGITAL Command Language (DCL) command LIBRARY (see Section 7.2) which you use to replace and maintain modules in an existing library, or to create a new library; and (2) a collection of Librarian routines (see Section 7.4) that you can call from a program to initialize and open a library, and to retrieve, insert, and delete modules.

7.1 LIBRARIES

The following sections describe the four library types -- object, macro, help, and text -- and the contents of library indexes.

7.1.1 Types of Libraries

There are four types of libraries, distinguished by their file types:

- Object libraries (file type OLB) contain frequently called routines and are used as input to the linker. The linker searches the object module library whenever it encounters a reference it cannot resolve from the specified input files. See the VAX-11 Linker Reference Manual for more information on how the linker uses libraries.

- Macro libraries (file type MLB) contain macro definitions used as input to the assembler. The assembler searches the macro library whenever it encounters a macro that is not defined in the input file. See the VAX-11 MACRO Language Reference Manual for information on defining macros.

- Help libraries (file type HLB) contain help modules; that is, modules that provide user information about a program. You can retrieve help messages in your program by calling the appropriate Librarian routines. See Section 7.3 for information about creating help modules for insertion into help libraries.

- Text libraries (file type TLB) contain any sequential record files that you want to retrieve as data for your program. Your programs can retrieve text from text libraries by calling the appropriate Librarian routines. See the /INSERT qualifier in Section 7.2.2 for information about inserting text files into text libraries.

You use DCL commands to manipulate libraries in their entirety; for example, the DELETE, COPY, and RENAME commands delete, copy, and rename libraries, respectively. For more information on file maintainance, see the VAX/VMS Command Language User's Guide.

7.1.2 Structure of Library Indexes

Every library contains a library header that describes the contents of the library. The information in the library header includes:

- The type of library
- The number of indexes and their location in the file
- The version number of the Librarian
- The library's creation date and time
- The last update date and time
- The library's preallocated size and its current size

Each module has a module header that contains information about the module, including its type, its attributes, and its date of insertion into the library.

Libraries can contain more than one index. All libraries contain an index for the module name table (MNT). Object module libraries also contain an index called a global symbol table (GST) that is a list of the global symbols defined in each of the library modules.

The MNT catalogs modules by module name, rather than by the name of the input file that contained the inserted module. The only exception to this procedure occurs with text libraries, for which the file name of the input file containing the text automatically becomes the module name. See the description of the /MODULE qualifier in Section 7.2.1.

7.2 THE DCL LIBRARY COMMAND

This section describes how to create, modify, and maintain libraries using the DCL command LIBRARY. This information also appears in the VAX/VMS Command Language User's Guide.

The purpose of the LIBRARY command is to maintain object, macro, help, or text libraries. The command's default operation is to replace modules. By specifying various qualifiers, you can also use the LIBRARY command to create and modify libraries; and to insert, delete, extract, and list library modules and symbols.

7.2.1 Library Command String

To use the LIBRARY command, enter the following command string in response to the DCL prompt:

```
$ LIBRARY/qualifier(s)  library-file-spec  [input-file-spec[/MODULE=module-name][,...]]
```

/qualifier(s)

The function(s) to be performed by the LIBRARY command. Section 7.2.2 describes the qualifiers in detail.

library-file-spec

The name of the library you want to create or modify. This parameter is required. If you do not specify a library file, you will be prompted for one as follows:

```
$_Library:
```

No wild card characters are allowed in the library file specification.

If the file specification or a qualifier in the command line does not include a file type, the LIBRARY command assumes a default type of OLB, indicating an object library.

NOTE

Any attempt to modify a library that was created by the VAX-11 Version 1.0 Librarian results in an automatic compression into the new format introduced with Version 2.0. The compression occurs before the requested modification. (See the /COMPRESS qualifier in Section 7.2.2.) Furthermore, libraries created before Version 2.0 that have not been modified or compressed will appear in a different format when listed by the /LIST qualifier.

input-file-spec[,...]

The names of one or more files that contain modules you want to insert into the specified library.

Whenever you include an input file specification, the LIBRARY command either replaces or inserts the modules contained in the input file(s) into the specified library. The input file specification is required when you specify either /REPLACE (the LIBRARY command's default operation) or /INSERT, which is an optional qualifier. If you do not specify an input file when you use these qualifiers, you will be prompted for it as follows:

```
$_File:
```

When you use the /CREATE qualifier to create a new library, the input file specification is optional. If you include an input file specification with /CREATE, the LIBRARY command first creates a new library, and then inserts the contents of the input file(s) into the library.

Note that the /EXTRACT qualifier does not accept an input file specification.

If you specify more than one input file, separate the file specifications with commas (,). The LIBRARY command will then insert the contents of each file into the specified library.

If any file specification does not include a file type and if the command string does not indicate one, the LIBRARY command will assume a default file type of OBJ, designating an object library. You can control the default file type by specifying the appropriate qualifier as indicated below.

Qualifier	Default File Type
/HELP	HLP
/MACRO	MAR
/OBJECT	OBJ
/TEXT	TXT

Note also that the file type you specify with the library file specification affects the default file type of the input file specification, provided that you do not specify the /CREATE qualifier. For example, if the library file type is HLB, MLB, OLB, or TLB, the input file type default will be HLP, MAR, OBJ, or TXT, respectively.

Wild card characters are allowed in the input file specification(s).

/MODULE=module-name

The module-name of a text module you want to insert or replace. When you are inserting text modules into a library, the input file that you specify is taken to be a single module. Therefore, the file name of the input file specification becomes the module-name. If you want the file you are inserting to have a module-name different from the input file name, use the /MODULE qualifier to name the added module.

You can also use the /MODULE qualifier to enter a text module interactively. If you specify the logical name SYS$INPUT as the input file, and issue the /MODULE qualifier, the LIBRARY command will insert the text you enter from the console into the specified library module. To terminate the console input, press <CTRL/Z>.

Remember that the /MODULE qualifier is an input file qualifier; it assumes that you are either replacing or inserting a new text module. Therefore, the qualifiers that remove modules -- /EXTRACT, /DELETE, /REMOVE -- are incompatible with /MODULE.

7.2.2 Command Qualifiers

When using the LIBRARY command, you can specify qualifiers that request more than one function in a single command, with some restrictions. Generally, you cannot specify multiple qualifiers that request incompatible functions. The qualifiers that perform library functions, related qualifiers, and qualifier incompatibilities are summarized in Table 7-1.

Table 7-1
LIBRARY Command Qualifier Compatibilities

Qualifier	Related Qualifiers	Incompatible Qualifiers
/COMPRESS	/OUTPUT	/CREATE, /EXTRACT
/CREATE [1]	/SQUEEZE,[2] /GLOBALS [3] /SELECTIVE_SEARCH [3]	/COMPRESS, /EXTRACT
/CROSS_REFERENCE	/ONLY	/EXTRACT
/DELETE	---	/EXTRACT
/EXTRACT	/OUTPUT	/COMPRESS, /CREATE, /DELETE, /INSERT /LIST, /REMOVE /REPLACE
/INSERT	/SQUEEZE,[2] /GLOBALS [3] /SELECTIVE_SEARCH [3]	/EXTRACT
/LIST	/FULL, /NAMES,[3] /ONLY	/EXTRACT
/REMOVE [3]	---	/EXTRACT
/REPLACE	/SQUEEZE,[2] /GLOBALS [3] /SELECTIVE_SEARCH [3]	/EXTRACT
/MODULE [4]	/TEXT	/EXTRACT, /DELETE /REMOVE

1. The /CREATE, /INSERT, and /REPLACE qualifiers are not incompatible; however, if you specify more than one, /CREATE takes precedence over /INSERT, and /INSERT takes precedence over /REPLACE. The related qualifiers for /CREATE are applicable only if you enter one or more input files.

2. This qualifier applies only to macro libraries.

3. This qualifier applies only to object libraries.

4. This file qualifier applies only to text libraries.

/COMPRESS[=(option[,...])]

Requests the LIBRARY command to perform either of the following functions:

- Recover unused space in the library resulting from module deletion
- Reformat a library created by the VAX/VMS Version 1.0 Librarian into the Version 2.0 format

When you specify /COMPRESS, the LIBRARY command by default creates a new library with a version number one higher than the existing library. Use the /OUTPUT qualifier to specify an alternate name for the compressed library.

Specify one or more of the following options to increase or decrease the size of the library, overriding the values specified when the library was created:

BLOCKS:n — Specifies the number of 512-byte blocks to be allocated for the library

GLOBALS:n — Specifies the maximum number of global symbols the library can contain (for object module libraries only)

MODULES:n — Specifies the maximum number of modules or macros the library can contain

KEYSIZE:n — Changes the maximum length of module names or global symbol names.

If you specify more than one option, separate them with commas and enclose the list in parentheses.

/CREATE[=(option[,...])]

Requests the LIBRARY command to create a new library. When you specify /CREATE, you can optionally specify a file or a list of files that contain modules to be placed in the library.

By default, the LIBRARY command creates an object module library; specify /MACRO, /HELP, or /TEXT to indicate that the library is a macro, help, or text library.

Specify one or more of the following options to control the size of the library, overriding the system defaults:

BLOCKS:n — Specifies the number of 512-byte blocks to be allocated for the library. By default, the LIBRARY command allocates 100 blocks for a new library.

GLOBALS:n — Specifies the maximum number of global symbols the library can contain initially. By default, the LIBRARY command sets a maximum of 128 global symbols for an object module library. (Macro, help, and text libraries do not have a global symbol directory; therefore, the maximum for these libraries defaults to 0.)

MODULES:n — Specifies the maximum number of modules the library can contain. By default, the LIBRARY command sets an initial maximum of 512 modules for an object module library and 256 modules for a macro, help, or text library.

KEYSIZE:n — Specifies the maximum name length of modules or global symbols. By default, the LIBRARY command limits the names of global symbols and object, macro, and text modules to 31 characters. The name length limit for help modules is 15 characters.

KEYSIZE:n (Cont.) When you specify a key-size value, remember that VAX-11 MACRO and the VAX-11 Linker do not accept module names or global symbol names in excess of 31 characters.

If you specify more than one option, separate them with commas and enclose the list in parentheses.

/CROSS_REFERENCE[=(option[,...])]

Requests a cross-reference listing of an object library.

If you omit this qualifier, cross-reference listings will not be provided. However, if you specify /CROSS_REFERENCE without specifying an option, you will obtain cross-reference listings by default that contain only symbols by name and symbols by value.

You can specify one or more of the following options:

ALL — Specifies that all types of cross references are required

MODULE — Specifies a cross-reference listing of both the global symbol references in the module and the global symbol definitions

NONE — Specifies that no cross-reference listing is required.

SYMBOL — Provides a cross-reference listing by symbol name

VALUE — Provides a cross-reference listing of symbols by value

If you specify more than one option, separate the options with commas and enclose the list in parentheses.

/DELETE=(module[,...])

Requests the LIBRARY command to delete (physically remove) one or more the modules from a library. You must specify the names of the modules to be deleted. If you specify more than one module, separate the module names with commas and enclose the list in parentheses.

Wild card characters are allowed in the module specification.

If you specify the /LOG qualifier with /DELETE, the LIBRARY command will issue the message:

```
%LIBRAR-S-DELETED, MODULE module-name DELETED FROM library-name
```

/EXTRACT=(module[....])

Copies one or more modules from an existing library into a new file. If you specify more than one module, separate the module names with commas and enclose the list in parentheses.

Wild card characters are allowed in the module specification.

If you specify the /OUTPUT qualifier with /EXTRACT, the LIBRARY command will write the output into the file specified by the /OUTPUT qualifier. If you specify /EXTRACT and do not specify /OUTPUT, the LIBRARY command will write the file into a file that has the same file name as the library and a file type of OBJ, MAR, HLP, or TXT depending on the type of library.

/FULL

Requests a full description of each module in the module name table. Use this qualifier with the /LIST qualifier to request a list of each library module in the format:

```
module-name [Ident  nn]dd   Inserted dd-mmm-yyyy   hh:mm:ss   [n symbols]
```

The identification number and the number of symbols appear only in object libraries.

/GLOBALS
/NOGLOBALS

Control, for object module libraries, whether the names of global symbols in modules being inserted in the library are included in the global symbol table.

By default, the LIBRARY command places all global symbol names in the global symbol table. Use /NOGLOBALS when you do not want the global symbol names in the global symbol table.

/HELP

Indicates that the library is a help library. When you use the /HELP qualifier, the library file type defaults to HLB and the input file type defaults to HLP.

/INSERT

Requests the LIBRARY command to add the contents of one or more files to an existing library. If an object module input file consists of concatenated object modules, the LIBRARY command will create a separate entry for each object module in the file; each module name table entry reflects an individual module name. If a macro or help file specified as input contains more than one definition, the LIBRARY command will create a separate entry for each one, naming the module name table entries according to the names specified on the .MACRO directives or in the key-1 name in the HELP format (see Section 7.3.1).

In text libraries, unlike object, macro, and help libraries, the input file contains data records of undefined contents. Therefore, the Librarian catalogs the entire input file as a single module using the input file specification as the module name. If you want to rename the inserted module, use the /MODULE qualifier described in Section 7.2.1.

When the LIBRARY command inserts modules into an existing library, it checks the module name table before inserting each module. If a module name or global symbol name already exists in the library, an error message will be issued and the module or symbol will not be added to the library.

To insert or replace a module in a library regardless of whether there is a current entry with the same name, use the /REPLACE qualifier (the default operation).

/LIST[=file-spec]
/NOLIST

Control whether the LIBRARY command creates a listing of the contents of the library.

By default, no listing is produced. If you specify /LIST without a file specification, the LIBRARY command will write the output file to the current SYS$OUTPUT device. If you include a file specification that does not have a file type, the LIBRARY command will use the default file type of LIS.

If you specify /LIST with qualifiers that perform additional operations on the library, the LIBRARY command will create the listing after completing all other requests; thus, the listing reflects the status of the library after all changes have been made.

When you specify /LIST, the LIBRARY command provides, by default, the following information about the library:

```
Directory of OBJECT library _DBB0:[LIBRAR]LIBRAR.OLB;1 on 14-NOV-1979 10:08:28
Creation date:  12-NOV-1979 19:40:36      Creator:  VAX-11 Librarian V01.02
Revision date:  14-NOV-1979 16:04:58      Library format:  1.1
Number of modules:     15                 Max. key length:  31
Other entries:         73                 Preallocated index blocks:     35
Recoverable deleted blocks:     15        Total index blocks used:       12
```

/LOG
/NOLOG

Control whether the LIBRARY command verifies each library operation. If you specify /LOG, the LIBRARY command will display the module name, followed by the library operation performed, followed by the library file specification. Examples of the /LOG qualifier appear in the descriptions of /DELETE and /REPLACE.

/MACRO

Indicates that the library is a macro library. When you specify /MACRO, the library file type defaults to MLB and the input file type defaults to MAR.

/NAMES
/NONAMES

Controls, when /LIST is specified for an object module library, whether the LIBRARY command lists the names of all global symbols in the global symbol table as well as the module names in the module name table.

The default is /NONAMES, which does not list the global symbol names. If you specify /NAMES, each module entry name will be displayed in the format:

```
module   "module-name"

global-symbol     global-symbol     global-symbol     global-symbol
      .                 .                 .                 .
      .                 .                 .                 .
      .                 .                 .                 .
```

If the library is a macro, help, or text library and you specify /NAMES, no symbol names will be displayed.

/OBJECT

Indicates that the library is an object module library. This is the default condition. The LIBRARY command assumes a library file type of OLB and an input file type of OBJ.

/ONLY=(module[....])

Specifies the individual modules on which the LIBRARY command can operate. When you use the /ONLY qualifier, the LIBRARY command lists or cross references only those modules specified.

If you specify more than one module, separate the module names with commas and enclose the list in parentheses.

Wild card characters are allowed in the module name specification.

/OUTPUT=file-spec

Specifies, when the /EXTRACT, /COMPRESS, or /CROSS_REFERENCE qualifiers are specified, the file specification of the output file.

For /EXTRACT, the output file contains the modules extracted from a library; for /COMPRESS, the output file contains the compressed library; for /CROSS_REFERENCE, the output file contains the cross-reference listing.

No wild card characters are allowed in the file specification.

If you omit the file type in the file specification, a default will be used depending on the library function qualifier and, in some cases, the library type qualifier as shown below.

Qualifier	Library Type Qualifier	Default File Type
/COMPRESS	/HELP	HLB
	/MACRO	MLB
	/OBJECT	OLB
	/TEXT	TLB
/CROSS_REFERENCE	---	LIS
/EXTRACT	/HELP	HLP
	/MACRO	MAR
	/OBJECT	OBJ
	/TEXT	TXT

/REMOVE=(symbol[,...])

Requests the LIBRARY command to delete one or more entries from the global symbol table in an object library. If you specify more than one symbol, separate the symbols with commas and enclose the list in parentheses.

Wild card characters are allowed in the symbol specification.

To display the names of the deleted global symbols, you must also specify the /LOG qualifier.

/REPLACE

Requests the LIBRARY command to replace one or more existing library modules with the modules specified in the input file(s). The LIBRARY command first deletes any existing library modules with the same name as the modules in the input file. Then, the new version of the module is inserted in the library. If any modules contained in the input file do not have a corresponding module in the library, the LIBRARY command will insert the new modules in the library.

This is the LIBRARY command's default operation. If you specify an input file parameter, the LIBRARY command will either replace or insert the contents of the input file into the library. If you use the /LOG qualifier with the /REPLACE qualifier, the LIBRARY command will display, in the following form, the names of each module that it replaces or inserts.

```
%LIBRAR-S-REPLACED, MODULE module-name REPLACED IN library-file-spec

%LIBRAR-S-INSERTED, MODULE module-name INSERTED IN library-file-spec
```

/SELECTIVE_SEARCH

Defines the input files being inserted into a library as candidates for selective searches by the linker. If you specify /SELECTIVE_SEARCH, the modules will be selectively searched by the linker when the library is specified as a linker input file: only the global symbol(s) in the module(s) referenced by other modules are included in the symbol table of the output image file.

/SQUEEZE
/NOSQUEEZE

Control whether the LIBRARY command compresses individual macros before adding them to a macro library. When you specify /SQUEEZE, which is the default, trailing blanks, trailing tabs, and comments are deleted from each macro before insertion in the library.

Use /SQUEEZE with the /CREATE, /INSERT, and /REPLACE qualifiers to conserve space in a macro library. If you want to retain the full macro, specify /NOSQUEEZE.

/TEXT

Indicates that the library is a text library. When you use the /TEXT qualifier, the library file type defaults to TLB and the input file type defaults to TXT.

/WIDTH=n

Controls the screen display width (in characters) for listing global symbol names. Specify the /WIDTH qualifier with the /NAMES qualifier to limit the line length of the /NAMES display.

The default display width is the width of the listing device. The maximum width is 132.

Examples

1. $ LIBRARY/CREATE TESTLIB ERRMSG,STARTUP

   The LIBRARY command creates an object module library named TESTLIB.OLB and places the modules ERRMSG.OBJ and STARTUP.OBJ in the library.

2. $ LIBRARY/INSERT TESTLIB SCANLINE
   $ LINK TERMTEST TESTLIB/LIBRARY

   The LIBRARY command adds the module SCANLINE.OBJ to the library TESTLIB.OLB. The library is specified as input to the linker by using the /LIBRARY qualifier on the LINK command. If the module TERMTEST.OBJ refers to any routines or global symbols not defined in TERMTEST, the linker will search the global symbol table of library TESTLIB.OLB to resolve the symbols.

3. $ LIBRARY/EXTRACT=(ALLOCATE,APPEND)/OUTPUT=MYHELP SYS$HELP:HELPLIB.HLB

   The LIBRARY command specifies that the modules ALLOCATE and APPEND be extracted from the help library HELPLIB.HLB and output to the file MYHELP.HLP.

4. $ LIBRARY/CROSS_REFERENCE=ALL/OUTPUT=SYS$OUTPUT LIBRAR

   The LIBRARY command requests a cross-reference listing of the object library LIBRAR.OLB. The cross-reference listing is output on the terminal. The listing includes cross-references by symbol, by value, and by module.

5. $ LIBRARY/REMOVE=(LIB_EXTRCT_MODS,LIB_INPUT_MAC)/LOG LIBRAR

   The LIBRARY command requests the removal of the global symbols LIB_EXTRCT_MODS and LIB_INPUT_MAC from the object library LIBRAR.OLB. The /LOG qualifier requests that the removal of the symbols be confirmed by messages.

6. $ LIBRARY/MACRO/CREATE=(BLOCKS:40,MODULES:100) MYMAC TEMP
   $ MACRO MYMAC/LIBRARY,CYGNUS/OBJECT

   The LIBRARY command creates a macro library named MYMAC.MLB from the macros in the file TEMP.MAR. The new library has room for 100 modules in a 40-block file. If the input file contains multiple macros, each macro will be entered in the new library.

   The MACRO command assembles the source file CYGNUS.MAR; the /LIBRARY qualifier specifies the library MYMAC.MLB as an input file. If the source file CYGNUS contains any macro calls not defined within the file, the assembler will search the library.

7. $ LIBRARY/LIST=MYMAC.LIS/FULL MYMAC.MLB

   The LIBRARY command requests a full listing of the macro library MYMAC; the output is written to a file named MYMAC.LIS.

8. $ LIBRARY/INSERT/TEXT TSTRING SYS$INPUT/MODULE=TEXT1

   The LIBRARY command inserts a module named TEXT1 into the text library TSTRING.TLB. The input is taken from SYS$INPUT.

9. $ LIBRARY/LIST/NAMES/ONLY=$ONE/WIDTH=80 SYMBOLIB

   The LIBRARY command requests a full listing of the module $ONE, contained in the object library SYMBOLIB.OLB. The /WIDTH qualifier requests that the global-symbol display be limited to 80 characters per line.

7.3 HELP LIBRARIES

Help messages are a convenient means of providing specific information about a program to an interactive user. The help messages are stored as modules in help libraries. Your programs can access the help modules by calling the appropriate Librarian routines described in Section 7.4. In this way, users of your program can quickly retrieve relevant information about using your program.

You create help libraries in the same manner that you create object, macro, and text libraries, using the LIBRARY/CREATE command described in Section 7.2.2. However, before you can insert modules into a help library, you must format the input file so that the Librarian can catalog its individual modules. This section describes how to create input files containing help modules.

7.3.1 Creating Help Files

The input file that you insert into a help library is a text file that you build with a text editor. Each input file may contain one or more help modules. A help module is a group of help messages that relates to the same topic, or key.

Each module within a help library contains a group of related key names, or topics, numbered key-1 through key-n. The key-1 name identifies the main topic of help information; for example, the name of a command in your program that requires explanation. The key-2 through key-n names identify subtopics that are related to the key-1 name; for example, the command's parameters and/or qualifiers. This organization enables users of your program to find a general message describing how to use the command, and then optionally to select subtopics that provide additional information about the command's parameters and qualifiers.

7.3.2 Formatting Help Files

Each key-1 line in the module consists of the key number (1) in the first column, followed by the name of the key. Subsequent subkey lines, key-2 through key-n, consist of the subkey number followed by the name of the subkey. For example, a help module for a command might have the following two key lines:

```
1 Command name
        .
        .
        .
   help message text
        .
        .
        .
2 Parameters
```

Each help source file can contain several modules. The Librarian recognizes an individual module as a group of key-1 and subkey lines, and their associated message text. A module is terminated either by another key-1 line or an end-of-file (EOF) record.

The format of a help source file is:

```
1 key-1 name
   .
   .
   .
  help message text
   .
   .
   .
2 key-2 name
   .
   .
   .
   help message text
   .
   .
   .
n key-n name
   .
   .
   .
1 key-1 name
```

The Librarian stores the key-1 name in its module name table; therefore, the name of the module is the same as the key-1 name. The subsequent numbers in the first column indicate that the line is a subkey. A module can have several subkeys with the same number. For example, a help module describing a command might have the following key-2 lines:

```
2 parameters
2 arguments
```

You can insert comments anywhere in a module. When the Librarian encounters an exclamation mark as the first character on a line, it assumes that the line is for comments. Comment lines that follow a key-1 line are included in the module. However, when your program retrieves help text, the Librarian does not output the comment lines.

The text of the help message may be any length; the only restriction to the text is that it cannot contain a number or a slash (/) in the first column of any line. A number in the first column of a line indicates that the line is a key. A slash (/) in the first column indicates a qualifier line.

A qualifier line is similar to a key line, except that the Librarian returns a list of all the qualifier lines when you request help either on a key-1 or on the key containing the qualifiers (usually a key-2 named "Qualifiers"). Therefore, if your help module describes a command that has qualifiers, the Librarian will provide a list of all the command's qualifiers whenever you request help on the command.

7.3.3 Help Message Example

The help module in Figure 7-1 shows the organization of help messages for the DCL LIBRARY command.

```
!  Description of DCL LIBRARY command
1 LIBRARY
Creates or modifies an object module library or a macro library; or
inserts, deletes, replaces, or lists modules, macros, or global symbol
names in a library.

Format
 LIBRARY library-file-spec [input-file-spec,...]
! This section lists the parameters to the LIBRARY command
2 Parameters
library-file-spec
 Specifies the name of the library to be created or modified
 Wild cards are not allowed in the library file specification. If
 the file specification does not include a file type, the LIBRARY
 command assumes a default file type of OLB if /OBJECT is
 specified either explicitly or by default; a default file
 type of MLB if /MACRO is specified; a default file type of HLB
 if /HELP is specified, or a default file type of TLB if /TEXT
 is specified.
input-file-spec,...
 Specifies the names of one or more files that contain modules to
 be inserted in the specified library.
 Wild cards are allowed in the input file specifications. If
 any file specification does not include a file type, the LIBRARY
 command assumes a default file type of OBJ when /OBJECT is
 specified either implicitly or by default; a file type of MAR
 when /MACRO is specified and /RSX11 is not specified; and a file
 type of MAC when /MACRO and /RSX11 are specified.
!This section lists the qualifiers to the LIBRARY command.
2 Qualifiers
/COMPRESS[=options,...]
 Requests  the LIBRARY command to recover unused space in the
 library resulting from module deletion.
 Options to override initial defaults for library:
 BLOCKS:n
 GLOBALS:n
 MODULES:n
/CREATE[=options,...]
 Requests the LIBRARY command to create a new library. When you
 specify /CREATE, you can also specify a file or a list of
 files that contain modules to be placed in the library.

.
.
.
```

Figure 7-1 Help Messages for LIBRARY Command

```
/GLOBALS (D)
/NOGLOBALS
Controls, for object module libraries, whether the
names of global symbols in modules being inserted
in the library are included in the global symbol
table.

By default, the LIBRARY command places all global
symbol names in the global symbol table.  Use
/NOGLOBALS when you do not want global
symbol names in the global symbol table.

.
.
.

/TEXT
 Indicates that the library is a text library.  The default
 library extension is TLB and the default file extension is
 TXT.
/WIDTH=n
 Specifies that the width of the listing of global symbols
 (requested by /NAMES) should be of the given width.  The Librarian
 will determine the width of the listing device if /WIDTH is
 not specified.
```

Figure 7-1 (Cont.) Help Messages for LIBRARY command

When you retrieve help messages, you specify the key-1 level, followed by any subkeys that contain appropriate help information. The Librarian returns the help message associated with the key path you specified.

To retrieve the LIBRARY command's key-1 help information, you would type the DCL command HELP LIBRARY. The Librarian would return the associated help message, followed by the message, "Additional information available:" and a list of all the key-2 names in the module. In this case, the Librarian also returns a list of all the qualifiers specified in the qualifier lines. Figure 7-2 displays the message returned from the HELP LIBRARY command.

```
LIBRARY

  Creates or modifies an object module library or a  macro  library;
  or inserts, deletes, replaces, or lists modules, macros, or global
  symbol names in a library.

  Format
   LIBRARY library [file-spec,...]

  Additional information available:

  Parameters Qualifiers
  /COMPRESS[=options,...]        /CREATE[=options,...] /CROSS[=option,...] (D=SYMBOL, VALUE)      /DELETE=module,...
  /EXTRACT=module,...   /FULL    /GLOBALS (D)          /NOGLOBALS          /HELP     /INSERT    /LIST[=file-spec]
  /NOLIST (D)           /LOG     /MACRO     /MODULE=module_name] /NAMES    /NONAMES (D)         /OBJECT (D)
  /ONLY=(module, module)         /OUTPUT=file-spec     /REMOVE=symbol,...   /REPLACE (D)        /RSX11     /SELECTIVE_SEARCH
  /SQUEEZE   /TEXT      /WIDTH=n
```

Figure 7-2 HELP LIBRARY Display

Note that you could not retrieve the key-2 level, "parameters," by typing HELP PARAMETERS. The Librarian searches for a subkey only after successfully finding the higher-level keys. In other words, if you want to retrieve a key-3 message, you would have to specify the key-1 and key-2 lines that are associated with the key-3 line.

Note also that if you request help information on the /GLOBALS qualifier, the Librarian wil return /NOGLOBALS as well. As shown in Figure 7-2, you can provide information on a qualifier that has more than one form by associating two qualifier lines with a single help message.

When the Librarian successfully searches the key path to the requested key, it displays all the key names in that path, followed by the help message associated with the last specified key. For example:

```
$HELP LIBRARY/HELP

        LIBRARY

           /HELP
            Indicates that the library is a HELP library. The
            default library file type is HLB and the input file type
            is HLP.
```

If you try to retrieve a help message that does not have a corresponding key in the module name table, the Librarian will issue a message. For example:

```
$HELP FIRE

        Sorry, no documentation on FIRE


        Additional information available:
```

This message will be followed by a list of all the module names in the module name table.

If you have correctly specified the key-1 line, but have requested a subkey that does not exist, the Librarian will print a message. For example:

```
        $HELP LIBRARY/FIRE

           Sorry, no documentation on LIBRARY/FIRE

           Additional information available:

           Parameters  Qualifiers
           /COMPRESS[=options...]       /CREATE[=options...]
                .
                .
                .
```

The message will include a list of all the subkeys associated with the last correctly specified key.

The help library serves as the repository for all your help messages. You can include help messages in your programs by calling the appropriate Librarian routines described in the next section.

7.4 LIBRARIAN ROUTINES

The Librarian provides a set of 18 routines that your programs can call to:

- Initialize a library
- Open a library
- Look up a key in a library
- Insert a new key in a library
- Return the names of the keys
- Delete a key and its associated text
- Read text records
- Write text records

Your programs can call the Librarian routines using the VAX-11 standard calling sequence provided in all languages that produce VAX-11 native-mode instructions. When your program calls the Librarian routines, it must furnish whatever arguments that the routine requires. When the routine completes execution, it returns control to your program. Your program should then analyze the success or failure of the requested operation. See Section 7.5 for an example of calling Librarian routines from a program.

When you link programs that contain calls to the Librarian routines, you must specify an options file to the input file parameter of the LINK command. The options file must contain the following file specification and qualifier:

```
SYS$LIBRARY:LBRSHR/SHARE
```

See Section 7.5 for an example of linking a program that references the Librarian routines.

For detailed information on linker option files, see the VAX-11 Linker Reference Manual.

Table 7-2 lists each Librarian routine and its function.

The following sections describe, in detail, each of the Librarian routines. The routines appear in alphabetical order. The order in which you call them in your program depends upon the library operations you need to perform. However, in all cases, you must call LBR$INI_CONTROL, followed by LBR$OPEN, before calling any other routine.

Each description of a routine provides the general format for calling the routine from your program. Spaces between arguments are included for readability, and are not part of the syntax. Following the format is a description of each of the arguments.

Each section lists the possible return status codes for the specific routine, with an explanation of the code. Success codes appear alphabetically before an alphabetical listing of the warning and error codes. For more information on return status codes, see Section 7.6.

In addition, each section provides further information, under "Notes," about the routine, including specific information about arguments. Any information that does not appear in another category appears under "Notes."

Table 7-2
Librarian Routines

Routine name	Function
LBR$INI_CONTROL	Initializes a library index for use by all other routines
LBR$OPEN	Opens an existing library or creates a new one
LBR$GET_HEADER	Retrieves information from the library header
LBR$CLOSE	Closes an open library
LBR$SET_INDEX	Sets the index number to be used during processing of the library
LBR$SET_MODULE	Reads, and optionally updates, a module header
LBR$LOOKUP_KEY	Looks up a key in the current index in preparation for reading the key's associated text
LBR$FIND	Looks up a key by its record identification in preparation for reading the key's associated text
LBR$INSERT_KEY	Inserts a new key in the current library index
LBR$REPLACE_KEY	Replaces an existing key in the current library index
LBR$DELETE_KEY	Deletes a key from a library index
LBR$DELETE_DATA	Deletes all the text records associated with a specified module
LBR$GET_RECORD	Reads a text record associated with a specified key
LBR$PUT_RECORD	Writes a text record to be associated with a specified key
LBR$PUT_END	Terminates a sequence of records written with LBR$PUT_RECORD
LBR$SEARCH	Finds index keys that point to specified text
LBR$GET_INDEX	Calls a user-supplied routine to return the contents of an index optionally qualified by a key
LBR$GET_HELP	Retrieves help text

LBR$CLOSE

7.4.1 LBR$CLOSE - Close a Library

The LBR$CLOSE routine closes an open library.

Format

LBR$CLOSE (library-index)

library-index

A pointer to a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

Return Status

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_ILLCTL

The specified library index is not valid.

Notes

If the library index is 0, LBR$CLOSE immediately returns with success.

Upon successful completion, LBR$CLOSE closes the open library, and deallocates all of the memory used for processing the library.

7.4.2 LBR$DELETE_DATA - Delete Text Records

The LBR$DELETE_DATA routine deletes all the text records associated with the specified module.

Format

LBR$DELETE_DATA (library-index, txtrfa)

library-index

A pointer to a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

txtrfa

A pointer to a 2-longword array that contains the record's file address (RFA) of the text you want to delete.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_INVRFA

The specified RFA is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_STILLKEYS

Keys in other indexes still point at the text; therefore, the specified text was not deleted.

Notes

If the reference count of the text is 0 (there are no other indexes pointing at the text), LBR$DELETE_DATA will delete the specified text records. If the reference count is not 0 (there are keys in other indexes pointing at the text), the Librarian returns the error LBR$_STILLKEYS.

The Librarian reuses data blocks that contain no text.

LBR$DELETE_KEY

7.4.3 LBR$DELETE_KEY - Delete a Key

The LBR$DELETE_KEY routine deletes a key from a library index.

Format

LBR$DELETE_KEY (library-index, key-name)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

key-name

A longword that contains one of the following:

1. The value of the key (for libraries with binary keys)
2. The address of a string descriptor for the key (for libraries with ASCII keys)

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_KEYNOTFND

The specified key has not been found.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_UPDURTRAV

The specified index update is not valid as an embedded routine.

Notes

If LBR$DELETE_KEY finds the key specified by key-name in the current index, it deletes the key.

You cannot call LBR$DELETE_KEY within the user-supplied routine specified in either the LBR$SEARCH or LBR$GET_INDEX routines.

7.4.4 LBR$FIND - Lookup a Key by its RFA

The LBR$FIND routine looks up a library key by its record's file address (RFA) and prepares to read the key's associated text.

Format

LBR$FIND (library-index, txtrfa)

library-index

A pointer to a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

txtrfa

A pointer to a 2-longword array that contains the RFA returned by the LBR$LOOKUP_KEY routine.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_ILLIDXNUM

The specified index number is not valid.

LBR$_INVRFA

The specified RFA is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

If the specified RFA is valid, LBR$FIND initializes internal tables so that you can read the associated text.

LBR$GET_HEADER

7.4.5 LBR$GET_HEADER - Retrieve Library Header Information

The LBR$GET_HEADER routine returns information from library's header to the caller.

Format

LBR$GET_HEADER (library-index, retary)

library-index

The pointer to a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

retary

An array of 128 longwords that receives the library header. The information in the returned array is shown in Table 7-3.

Table 7-3
Library Header Information Array Offsets

Offset in Longwords	Symbolic Name	Contents
0	LHI$L_TYPE	Library type
1	LHI$L_NINDEX	Number of indexes
2	LHI$L_MAJORID	Library format major identification
3	LHI$L_MINORID	Library format minor identification
4	LHI$T_LBRVER	ASCIC version of Librarian
12	LHI$L_CREDAT	Creation date/time
14	LHI$L_UPDTIM	Date/time of last update
16	LHI$L_UPDHIS	Virtual Block Number (VBN) of start of update history (reserved)
17	LHI$L_FREEVBN	First logically deleted block
18	LHI$L_FREEBLK	Number of deleted blocks
19	LHI$B_NEXTRFA	Record's File Address (RFA) of end of library
21	LHI$L_NEXTVBN	Next VBN to allocate at end of file
22	LHI$L_FREIDXBLK	Number of free pre-allocated index blocks

(continued on next page)

Table 7-3 (Cont.)
Library Header Information Array Offsets

Offset in Longwords	Symbolic Name	Contents
23	LHI$L_FREEIDX	Listhead for pre-allocated index blocks
24	LHI$L_HIPREAL	VBN of highest pre-allocated block
25	LHI$L_IDXBLKS	Number of index blocks in use
26	LHI$L_IDXCNT	Number of index entries (total)
27	LHI$L_MODCNT	Number of entries in index 1 (module names)
28	LHI$L_MHDUSZ	Number of bytes of additional information reserved in module header
29-128		Reserved to DIGITAL

Return Status

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_ILLCTL

The specified library index is not valid.

Notes

Upon successful completion, LBR$GET_HEADER places the library header information into the array.

LBR$GET_HELP

7.4.6 LBR$GET_HELP - Return Help Text

LBR$GET_HELP returns help text in a help library to the calling program.

Format:

LBR$GET_HELP (library-index, [line-width], [routine], [data], key-1, key-2,...,key-n)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

line-width

The address of a longword that contains the width of the listing line.

routine

The address of specified routine to call for each line of text you want output.

data

The address of a longword of data to pass to the routine specified in the routine argument.

key-1,key-2,...,key-n

The address(es) of one or more string descriptors for the key(s) that define the text to be output.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_NOTHLPLIB

The specified library is not a help library.

Notes

The optional line-width argument controls the width of the listing line when available help topics are printed. If you do not supply a line-width, or if you specify 0, the line-width defaults to 80 characters per line.

If you do not supply a routine argument, LBR$GET_HELP calls the Run-Time Library procedure LIB$PUT_OUTPUT to send the help text lines to the current output device (SYS$OUTPUT). However, if you want SYS$OUTPUT for you program to be a disk file, rather than the terminal, it is recommended that you supply a routine to output the test.

If the key-1 descriptor is 0, or if it is not present, LBR$GET_HELP will assume that the key-1 name is "HELP," and it ignores all the other keys. For key-2 through key-n, a descriptor address of 0, or a length of 0, or a string address of 0 will terminate the list.

The key argument may contain any of the following special character strings:

String	Meaning
*	Return all first-level help text in the library
KEY...	Return all help text associated with the specified key and its subkeys
*...	Return all help text in the library

LBR$GET_HELP returns all help text in the same format as the output returned by the DCL command HELP. If you do not want the help text indented to the appropriate help level, you must supply your own routine to change the format.

The routine that you specify to output each help text line contains an argument list of four longwords:

1. The first argument contains the address of a string descriptor for the line to be output.

2. The second argument contains the address of a longword that points to one or more flag bits. The flags describe the contents of the text being passed. The possible flags are:

   HLP$M_NOHLPTXT - The specified help text cannot be found.
   HLP$M_KEYNAMLIN - The text contains the key names of the printed text.
   HLP$M_OTHERINFO - The text is part of the information provided on additional help available.

   Note that if no flag bit is set, help text is being passed.

3. The third argument is the address specified in the data argument in LBR$GET_HELP (or the address of a 0 constant if no argument has been supplied).

4. The fourth argument is the address of a longword containing the current key level.

The routine that you specify must return with success or failure status. A failure status (low bit = 0) terminates the current call to LBR$GET_HELP.

LBR$GET_INDEX

7.4.7 LBR$GET_INDEX - Return the Contents of an Index

LBR$GET_INDEX calls a user-supplied routine to return the contents of an index optionally qualified by a key.

Format:

LBR$GET_INDEX (library-index, index-number, routine-name, [match-desc])

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

index-number

A pointer to a longword that contains the number of the primary index you want to return.

routine-name

The name of a routine that you supply to be called for each element in the index.

match-desc

The address of a string descriptor that identifies selected entries.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_ILLIDXNUM

The specified index number is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_NULIDX

The specified index is empty.

Notes

LBR$GET_INDEX calls with two arguments the routine you specified in the routine-name argument. The two arguments are:

1. Either address of a string descriptor for the entry (for libraries with ASCII keys) or the address of a value (for libraries with binary keys)

2. Address of a 2-longword array containing the entry's RFA

If the routine returns a false value (low bit = 0), LBR$GET_INDEX stops searching the index.

Note that the string descriptor passed to your routine is valid only for the duration of the supplied routine. If you need to use the string descriptor in later processing, you must first copy the string.

If you include the match-desc argument, LBR$GET_INDEX supplies only entries that match the specified string. The string can contain embedded asterisks (*) and percent signs (%) that serve as wild card characters in the string description. If you do not supply the match-desc argument, LBR$GET_INDEX uses an asterisk and matches all entries. The match-desc argument is supported only in libraries with ASCII keys.

The routine that you specify in routine-name cannot contain calls to either the LBR$DELETE or LBR$INSERT_KEY routine.

LBR$GET_RECORD

7.4.8 LBR$GET_RECORD - Read a Text Record

The LBR$GET_RECORD routine returns the next text record associated with a key.

Format

LBR$GET_RECORD (library-index, inbufdes [, outbufdes])

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open and the LBR$LOOKUP_KEY routine must have been called.

inbufdes

A pointer to a string descriptor for the user-supplied buffer.

outbufdes

A pointer to a string descriptor for the actual record returned.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_LKPNOTDON

The requested key lookup has not been done.

RMS$_EOF

An attempt has been made to read past the logical end of text.

Notes

Before calling LBR$GET_RECORD, you must first find the key by calling LBR$LOOKUP_KEY or LBR$FIND. When you call LBR$GET_RECORD, the Librarian fills the input buffer (described by inbufdes) with the text record. If you have optionally specified the output buffer string descriptor (outbufdes), the Librarian fills it with the actual length and address of the data.

7.4.9 LBR$INI_CONTROL - Initialize a Library Index

The LBR$INI_CONTROL routine initializes a library index for use by all other Librarian routines. You must call this routine before calling any other Librarian routine.

Format

LBR$INI_CONTROL (library-index, func, [type, namblk])

library-index

A pointer to a longword that will receive the index for the library.

func

The address of a longword that contains the library function to be performed. Valid functions are LBRC_CREATE, LBRC_READ, and LBR$C_UPDATE.

type

The address of a longword that contains the library type. If you specify a library type, LBR$OPEN will check for the correct library type.

namblk

The address of a VAX-11 Record Management Services (VAX-11 RMS) NAM block. If the NAM block has a file identification in it because it was used before, the Librarian will use the VAX-11 RMS open-by-NAM block option. The Librarian will fill in the information in the NAM block so that it can be used at a later time to open the library. This argument is optional and should be used if the library will be opened many times during a single run of the program. For a detailed description of VAX-11 RMS NAM blocks, see the VAX-11 Record Management Services Reference Manual.

Return Status

LBR$_NORMAL

The library index was initialized successfully.

LBR$_ILLFUNC

The function requested is not valid.

LBR$_ILLTYP

The specified library type is not valid.

LBR$_INVNAM

The specified VAX-11 RMS NAM block is not valid.

LBR$_TOOMNYLIB

An attempt was made to allocate more than 16 control indexes; 16 is the maximum allowed.

Notes:

After you initialize the library index, you must open the library, or create a new one using the LBR$OPEN routine. You can then call other Librarian routines that you need. Once you have completed working with a library, close it with the LBR$CLOSE routine.

LBR$INI_CONTROL initializes a library by filling the longword referenced by the library-index argument with the index of the library. Upon completion of the call, the index can be used to refer to the current library in all future routine calls. Therefore, your program must not alter this value.

7.4.10 LBR$INSERT_KEY - Insert a New Key

The LBR$INSERT_KEY routine inserts a new key in the current library index.

Format

LBR$INSERT_KEY (library-index, key-name, txtrfa)

library-index

A pointer to a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

key-name

A longword that contains one of the following:

1. The address of the value of the key (for libraries with binary keys)
2. The address of a string descriptor for the key (for libraries with ASCII keys)

txtrfa

A pointer to a 2-longword array that contains the RFA of the associated text. You must use the RFA returned by the first call to the LBR$PUT_RECORD routine.

Return Status

LBR$_IDXFUL

The specified index is full.

LBR$_ILLCTL

The specified library index is not valid.

LBR$_INVRFA

The specified RFA does not point to valid text.

LBR$_KEYINDEX

The index already contains the specified key.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_UPDURTRAV

The specified index update is not valid as an embedded routine.

Notes

You cannot call LBR$INSERT_KEY within the user-supplied routine specified in either the LBR$SEARCH or LBR$GET_INDEX routines.

LBR$LOOKUP_KEY

7.4.11 LBR$LOOKUP_KEY - Lookup a Library Key

The LBR$LOOKUP_KEY routine looks up a key in the library's current index and prepares to read the text associated with the key.

Format

LBR$LOOKUP_KEY (library-index, key-name, txtrfa)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

key-name

A longword that contains one of the following:

1. The address of the value of the key (for libraries with binary keys)

2. The address of a string descriptor for the key (for libraries with ASCII keys)

txtrfa

A pointer to a 2-longword array that receives the RFA of the text you want to read.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_KEYNOTFND

The specified key was not found.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

If LBR$LOOKUP_KEY finds the specified key, it initializes internal tables so that you can read the associated text.

The Librarian returns the RFA (consisting of the VBN and the byte offset) to the 2-longword array pointed to by txtrfa. Note that the array contains an RFA of only 48 bits.

7.4.12 LBR$OPEN - Open a Library

The LBR$OPEN routine opens an existing library or creates a new one. This routine must be called after you call LBR$INI_CONTROL and before you call any other Librarian routine.

Format

LBR$OPEN (library-index [,fns, create-options, dns, rlfna, rns, rnslen])

library-index

A pointer to a longword that contains the index returned by LBR$INI_CONTROL.

fns

The address of a string descriptor for the file name string. This argument must be included unless the VAX-11 RMS NAM block address was previously supplied in the LBR$INI_CONTROL routine and it contained a file identification. Otherwise, an error (LBR$_NOFILNAM) will result.

create-options

If you are creating a library with LBR$C_CREATE, you must include the create-options argument. The create-options argument is an array of 20 longwords that describes the characteristics of the library you want to create. Table 7-4 shows the entries that the array must contain.

dns

The address of a string descriptor for the default file name string.

rlfna

The address of a VAX-11 RMS NAM block for the related file name. If you do not specify rlfna, no related file name processing occurs. See the VAX-11 Record Management Services Reference Manual for details on processing related file names.

rns

The address of a string descriptor for the resultant file name string. If an error occurs during an attempt to open the library, the expanded name string will be returned.

rnslen

A pointer to a longword that receives the length of the resultant file name string (or the length of the expanded name string if there was an error in opening the library).

Table 7-4
Create-Options Array

Offset in Longwords	Symbolic Name		Contents
0	CRE$L_TYPE		Library type
	LBR_$C_TYP_UNK (0)		Unknown/unspecified
	LBR_$C_TYP_OBJ (1)		Object and/or shareable image
	LBR_$C_TYP_MLB (2)		Macro
	LBR_$C_TYP_HLP (3)		Help
	LBR_$C_TYP_TXT (4)		Text
	(5-127)		Reserved to DIGITAL
	LBR_$C_TYP_USR (128-255)		User-defined
1	CRE$L_KEYLEN	0 non-0	32-bit unsigned keys Maximum length of ASCII keys
2	CRE$L_ALLOC	non-0	Initial library file allocation
3	CRE$L_IDXMAX		Number of primary indexes (maximum of 8)
4	CRE$L_UHDMAX		Number of additional bytes to reserve in module header
5	CRE$L_ENTALL		Number of index entries to pre-allocate
6-20			Reserved to DIGITAL

Return Status

LBR$_OLDLIBRARY

The specified Version 1.0 library has been opened.

LBR$_ILLCREOPT

The requested create options are not valid or not supplied.

LBR$_ILLCTL

The specified library index is not valid.

LBR$_ILLFMT

The specified library format is not valid.

LBR$_ILLFUNC

The specified library function is not valid.

LBR$_INSVIRMEM

No virtual memory is available for the specified function.

LBR$_LIBOPN

The specified library is already open.

LBR$_NOFILNAM

The fns argument was not supplied, or the VAX-11 RMS NAM block was not filled in.

LBR$_OLDMISMCH

The library function requested conflicts with the old library type specified.

LBR$_TYPMISMCH

The library type requested conflicts with the library type specified.

Notes

When the library is successfully opened, the Librarian reads the library header into memory, sets the default index to 1, and updates the library index.

If the library cannot be opened because it is already open for a write operation, LBR$OPEN will retry the open operation every one second for a maximum of 30 seconds before returning the VAX-11 RMS error, RMS$_FLK, to the caller.

LBR$PUT_END

7.4.13 LBR$PUT_END - Terminate a Text Sequence Written to a Library

The LBR$PUT_END routine terminates a text sequence written to a library by the LBR$PUT_RECORD routine.

Format

LBR$PUT_END (library-index)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

Call LBR$PUT_END after you have written text records to the library with the LBR$PUT_RECORD routine. LBR$PUT_END terminates a text sequence by attaching a three-byte logical end-of-file record (hexadecimal 77,00,77) to the text.

LBR$PUT_RECORD

7.4.14 LBR$PUT_RECORD - Write a Text Record

The LBR$PUT_RECORD routine writes a text record beginning at the next free location in the library.

Format

LBR$PUT_RECORD (library-index, bufdes, txtrfa)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

bufdes

A pointer to a string descriptor that contains the buffer to receive the record output.

txtrfa

A pointer to a 2-longword array that receives the RFA of the newly created module header.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

If this is the first call to LBR$PUT_RECORD, the Librarian first writes a module header and returns its RFA to the 2-longword array pointed to by txtrfa. LBR$PUT_RECORD then writes the supplied text record to the library.

LBR$REPLACE_KEY

7.4.15 LBR$REPLACE_KEY - Change Text Pointer or Insert New Key

The LBR$REPLACE_KEY routine inserts a new key in an index by changing the pointer associated with an existing key, or by inserting a new key.

Format

LBR$REPLACE_KEY (library-index, key-name, oldrfa, newrfa)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

key-name

A longword that contains one of the following:

1. The address of the key's value (for libraries with binary keys)
2. The address of a string descriptor for the key (for libraries with ASCII keys)

oldrfa

A pointer to a 2-longword array that contains the RFA of the old text.

newrfa

A pointer to a 2-longword array that contains the RFA of the new text.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

LBR$_INVRFA

The specified RFA is not valid.

Notes

If LBR$REPLACE_KEY does not find the key in the current index, it calls the LBR$INSERT_KEY routine to insert the key.

If LBR$REPLACE_KEY finds the specified key, it performs the following:

1. Decreases by 1 the reference count for the old text (pointed to by oldrfa)

2. Increases by 1 the reference count for the new text (pointed to by newrfa)

3. Modifies the entry for the key so that it now points to the new text

LBR$SEARCH

7.4.16 LBR$SEARCH - Search an Index

The LBR$SEARCH routine finds index keys that point to the specified text.

Format

LBR$SEARCH (library-index, index-number, rfa-to-find, routine-name)

library-index

A pointer to a longword that contains the index returned by the LBR$INI_CONTROL routine. The library must be open.

index-number

A pointer to a longword that contains the number of the primary index you want to search.

rfa-to-find

A pointer to a 2-longword array that contains the RFA of the key you want to find.

routine-name

The name of a routine that you supply to call for each key containing the matching RFA.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_ILLIDXNUM

The specified index number is not valid.

LBR$_KEYNOTFND

The Librarian did not find any keys with the specified RFA.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

Use LBR$SEARCH to find index keys that point to some specified text. For example, you can call LBR$SEARCH to find all the global symbols associated with an object module in an object library.

If LBR$SEARCH finds an index key, it calls a user-supplied routine with two arguments. The two arguments are:

1. Either the address of a string descriptor for an ASCII key or the address of the value of a binary key

2. Address of a 2-longword array that points to the RFA of the associated text

If the specified routine returns a false value (low bit = 0), then the index search terminates.

Note that the key-name argument is valid only for the duration of the call to the user-supplied routine. If you want to use the key-name argument later, you must copy it.

The routine that you specify in routine-name cannot contain any calls to either the LBR$DELETE or LBR$INSERT_KEY routine.

LBR$SET_INDEX

7.4.17 LBR$SET_INDEX - Set the Primary Index Number

The LBR$SET_INDEX routine sets the index number to use during processing of libraries that have more than one index.

Format

LBR$SET_INDEX (library-index, index-number)

library-index

A pointer to a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

index-number

A pointer to a longword that contains the number of the index you want to set.

Return Status

LBR$_ILLCTL

The specified library index is not valid.

LBR$_ILLIDXNUM

The index number specified is not valid.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

You call LBR$SET_INDEX when working with libraries that contain more than one index. Macro, help, and text libraries contain only one index; therefore, you do not need to call LBR$SET_INDEX. Object libraries contain two indexes. If you want to access the global symbol table, you must call the LBR$SET_INDEX routine to set the index number. User-developed libraries can contain more than one index; therefore, you may need to call LBR$SET_INDEX to set the index number.

Upon successful completion, LBR$SET_INDEX sets the current index to the requested index number. The Librarian numbers indexes starting with 1.

7.4.18 LBR$SET_MODULE - Read or Update a Module Header

The LBR$SET_MODULE routine reads, and optionally updates, the module header associated with a given record's file address (RFA).

Format

LBR$SET_MODULE (library-index, rfa, [bufdesc,buflen,updatedesc])

library-index

The address of a longword that contains the library index returned by the LBR$INI_CONTROL routine. The library must be open.

rfa

A pointer to the RFA associated with the module header. The Librarian returns the RFA as a result of either the first call to the LBR$PUT_RECORD routine, or a previous call to the LBR$LOOKUP_KEY routine. For a description of record access by RFA, see the VAX-11 Record Management Services Reference Manual.

bufdesc

A pointer to a string descriptor for the buffer that receives the module header.

buflen

The address of a longword to contain the length of the returned module header.

updatedesc

A pointer to a string descriptor of additional data that the Librarian stores with the module header. If you include this argument, the Librarian will update the module header with the additional information.

Return Status

LBR$_HDRTRUNC

The buffer supplied to hold the module header was too small.

LBR$_ILLCTL

The specified library index is not valid.

LBR$_ILLOP

The updatedesc argument was supplied and the library was a Version 1.0 library or the library was opened only for read access.

LBR$_INSVIRMEM

No virtual memory is available for the specified function.

LBR$_INVRFA

The specified RFA does not point to a valid module header.

LBR$_LIBNOTOPN

The specified library is not open.

Notes

If you specify bufdesc, the Librarian will return the module header into the buffer. If you specify buflen, the Librarian will also return the buffer's length. If you specify updatedesc, the Librarian will update the header information.

You define the maximum length of the update information when you create the library. The Librarian will zero-fill the information if it is less than the maximum length, or truncate the information if it exceeds the maximum length.

7.5 EXAMPLE OF LIBRARIAN ROUTINES

This section describes LBRDEMO, a VAX-11 FORTRAN program example that illustrates the use of the Librarian routines. LBRDEMO contains calls to the Librarian routines that perform the following operations:

- Name and initialize a text library
- Open or create a text library
- Replace or insert text modules
- Extract text modules
- List the help topics in the system help library
- Retrieve help text from the system help library

The sample program LBRDEMO is shown below. The circled numbers in the program are keyed to the descriptions that follow.

```
.TITLE DEMOMAC

;
; macros
;
        $credef                                 ; define create options array offsets
        $dscdef                                 ; define string descriptor offsets
        $lbrdef                                 ; define librarian parameters
        $lbrctltbl                              ; define library control table offsets
        $namdef                                 ; define NAM block offset
;
; set up FORTRAN COMMON block to allow FORTRAN main program to (1)
; access librarian data
;
        .PSECT  lbrdata, PIC, OVR, REL, GBL, SHR, NOEXE, RD, WRT, LONG

        .long   lbr$c_read                      ; func_read
        .long   lbr$c_create                    ; func_create
        .long   lbr$c_update                    ; func_update
        .long   lbr$c_typ_txt                   ; type_text
        .long   lbr$c_typ_hlp                   ; type_help
        .long   rms$_eof                        ; rmseof
        .long   dsc$k_class_d                   ; class_dynamic

                ; offsets into create options array:
                ; values are divided by 4 to convert byte
                ; offsets into longword offsets
        .long   cre$l_type/4                    ;  type of library
        .long   cre$l_keylen/4                  ;  max key length
        .long   cre$l_alloc/4                   ;  initial library disk allocation
        .long   cre$l_idxmax/4                  ;  number of indices
        .long   cre$l_uhdmax/4                  ;  size of additional module header data
        .long   cre$l_entall/4                  ;  number of index entries to preallocate

        .SBTTL  NAM_INIT - initialize RMS NAM block

;++
;       initialize array to be an RMS NAM block (2)
;
; Calling sequence:
;
;       Call nam_init (nam_array, result_desc)
;
; Inputs:
;
```

```
;       nam_array       Address of array of 56 bytes to be initialized
;                       as a NAM block
;
;       result_desc     Address of string descriptor for resultant name
;                       string.
;
; Outputs:
;
;       the nam_array is initialized as a NAM block, with the expanded
;       and resultant name strings pointing to the string described by
;       result_desc.
;
; Routine value:
;
;       Always success
;
;--


        .PSECT  $code$, PIC, REL, SHR, EXE, RD, NOWRT

        .ENTRY nam_init,^mR2, R3, R4, R5, R6>

        movl    4(AP), R6                               ; Get address of NAM block
        movc5   #0, (SP), #0, #nam$c_bln, (r6)          ; Zero the NAM block
        movl    8(AP), R0                               ; Get address of resultant name string descriptor
        $NAM_STORE NAM = R6,-                           ; Initialize the NAM fields
                        BLN = #nam$c_bln,-              ;  block length
                        BID = #nam$c_bid,-              ;  block id
                        RSS = dsc$w_length(R0),-        ; resultant name string size
                        ESS = dsc$w_length(R0),-        ; expanded name string size
                        RSA = @dsc$a_pointer(R0),-      ; resultant name string address
                        ESA = @dsc$a_pointer(R0),-      ; expanded name string address
        movl    #1,r0                                   ; return with success
        ret

        .SBTTL  retrieve_rmsstv

;       Retrieve the RMS STV returned from the last Librarian operation ❸
;
; Calling sequence:
;
        istv = retrieve_rmsstv (0)
;
; Inputs:
;
;       NONE
;
; Implicit inputs:
;
;       The library must be open.
;
; Outputs:
;
;       NONE
;
; Routine value:
;
;       The RMS STV from the last librarian operation
;
;--

        .ENTRY  retrieve_rmsstv,^M<>

        movl    g|Lbr$gl_rmsstv,r0      ;get the STV value
        ret                             ;and return with that as the function value
```

```
        .SBTTL  set_locate_mode

;++
;       Set the current library to read in locate mode
;
; Calling sequence:
;
;       call    set_locate_mode
;
; Inputs:
;
;       NONE
;
; Implicit inputs:
;
;       An operation must have been done on the desired library
;       (i.e. LBR$OPEN) before calling this routine.
;
; Outputs:
;
;       NONE
;
;--

        .ENTRY  set_locate_mode,^M<>

        movl    g^lbr$gl_control,r0             ;get address of control block
        bisl2   #lbr$m_locate,lbr$l_usrflg(r0)  ;set locate mode
        movl 1,R0                               ;return success
        ret

        .END
```

```
C
C       Demo program for library access procedures
C
        PROGRAM LBRDEMO

        IMPLICIT INTEGER (A-Z)
        EXTERNAL list_module
C
C       The common block is declared in a MACRO source module to gain
C       access to system definitions not available to FORTRAN.
C
        COMMON /lbrdata/ func_read, func_create, func_update,
        1 type_text, type_help, rmseof, class_dynamic, create_type,
        2 create_keylen, create_alloc, create_idxmax, create_uhdmax,❶
        3 create_entall

        CHARACTER*128 library_name, library_rsn, module_name
        CHARACTER*128 input_line
        CHARACTER*32 key1, key2, key3

        BYTE help_namblk (56), string_desc_bytes (8), dyn_desc_bytes (8)

        DIMENSION create_options (0:49), old_module_rfa (2), module_rfa (2),
        1 dyn_string (2), string_desc (2)

C
C       The equivalence of the STRING_DESC array with the STRING_DESC_BYTES
C       array is done to access the string descriptor class field.
C
```

```
        EQUIVALENCE (string_desc, string_desc_bytes),
        1 (dyn_string, dyn_desc_bytes)

        library_open = .false.
        have_name = .false.
C
C       Initialize NAM block for use with HELP library
C
        CALL nam_init (help_namblk, library_rsn) ❷
C
C       Allocate a dynamic string and initialize string descriptors
C
        dyn_string (1) = 0
        dyn_string (2) = 0
        dyn_desc_bytes (4) = class_dynamic
        string_desc (1) = 0
        string_desc (2) = 0
        string_desc_bytes (4) = class_dynamic
        status =
        1 lib$sgetl_dd (2048, string_desc)      !allocate 2048 byte string
        IF (status) GOTO 100
 10     CALL lib$signal (%VAL (status))
        STOP
C
C       Main dispatch loop -- Get action and dispatch ❹
C
 100    TYPE 9000
        READ (5, *, END=300) action
        GOTO (1000, 2000, 3000, 4000, 5000, 6000, 7000), action + 1
        GOTO 100
 200    CALL lib$signal (%VAL (status))
        GOTO 100
C
C       Close library and exit
C
 300    IF (library_open) THEN
                status = lbr$close (text_index)
                IF (.NOT. status) GOTO 10
                END IF
        STOP

C
C       Give some help
c
 1000   TYPE 9020
        GOTO 100
C
C       Name new library
C
C       If there is a library open, it will be closed.  The new library name
C       is accepted.
C
 2000   IF (library_open) THEN ❺
                status = lbr$close (text_index)
                IF (.NOT. status) CALL lib$signal (%VAL (status))
                library_open = .false.
                END IF
        TYPE 9040
 2050   READ (5, 9110, END=100) name_length, library_name
        library_name = library_name (1:name_length)//'.TLB'
        have_name = .true.
        GOTO 100
C
C       Open or Create a TEXT library ❻
C
 3000   IF (.NOT. have_name) GOTO 8000
```

```
 3010   TYPE 9540
 3020   READ (5, *, END = 100) create_flag
 3050   IF (create_flag) THEN
            TYPE 9060
            READ (5, *, END=100) max_key_length ❼
            function = func_create
            create_options (create_type) = type_text
            create_options (create_keylen) = max_key_length
            create_options (create_alloc) = 100
            create_options (create_idxmax) = 1
            create_options (create_uhdmax) = 0
            create_options (create_entall) = 100
            ELSE
            function = func_update
            END IF
C
C       Initialize librarian for this library
C
        status = lbr$ini_control (text_index, function, type_text)
        IF (.NOT. status) GOTO 200
C
C       Open or create the library
C
 3070   status = lbr$open (text_index, library_name, create_options) ❸
        IF (.NOT. status) GOTO 200
        library_open = .TRUE.

C       Note: if using locate mode for record transfer, call set_locate_mode here.

        GOTO 100
C
C       Insert or replace a module in the text library
C
 4000   IF (.NOT. library_open) GOTO 8020 ❽
        TYPE 9080
 4020   READ (5, 9110, END=100) name_length, module_name
        replacing = lbr$lookup_key (text_index,
        1 module_name (1:name_length), old_module_rfa)
        TYPE 9100
 4100   READ (5, 9110, END = 4200) line_length, input_line
        status = lbr$put_record (text_index,
        1 input_line (1:line_length), module_rfa)
 4120   IF (.NOT. status) CALL lib$signal (%VAL (status))
        GOTO 4100
C
C       Module text has been inserted into the library. Terminate the module
C
 4200   status = lbr$put_end (text_index)
        IF (.NOT. status) CALL lib$signal (%VAL (status)) ❾
        status = lbr$replace_key (text_index,
        1 module_name (1:name_length), old_module_rfa, module_rfa)
        IF (.NOT. status) CALL lib$sigal (%VAL (status))
        status = .TRUE.
        IF (replacing) status = lbr$delete_data (text_index,
        1 old_module_rfa)
        IF (.NOT status) GOTO 200
        GOTO 100
C
C       Extract module from library and type on terminal
C
 5000   IF (.NOT. library_open) GOTO 8020 ❿
        TYPE 9400
        READ (5, 9050, END = 100) module_name
        status = lbr$lookup_key (text_index, module_name, module_rfa)
        IF (.NOT. status) GOTO 200
 5100   status = lbr$get_record (text_index, string_desc, dyn_string)
```

```
        IF ((.NOT. status) .AND. status .NE. rmseof)
        1 CALL lib$signal (%VAL (status))
        IF (status .EQ. rmseof) GOTO 100
        CALL lib$put_output (dyn_strin)
        GOTO 5100
C
C       List contents of index of SYS$HELP:HELPLIB.HLB
C
 6000   status = lbr$ini_control (help_index, func_read, type_help)❶❶
        IF (.NOT. status) GOTO 200
        status = lbr$open (help_index, %descr ('SYS$HELP:HELPLIB.HLB'))
        IF (.NOT. status) GOTO 200
        status = lbr$get_index (help_index, 1, list_module)
        IF (.NOT. status) CALL lib$signal (%VAL (status))
        status = lbr$close (help_index)
        IF (.NOT. status) GOTO 200
        GOTO 100
C
C       Lookup help text in SYS$HELP:HELPLIB.HLB and display on the terminal
C
 7000   TYPE 9200
        READ (5, 9110, END = 100) KEY1LEN, KEY1 ❶❷
        IF (KEY1LEN .EQ. 0) GOTO 7020
        TYPE 9220
        READ (5, 9110, END = 100) KEY2LEN, KEY2
        IF (KEY2LEN .EQ. 0) GOTO 7020
        TYPE 9240
        READ (5, 9110, END = 100) KEY3LEN, KEY3
 7020   status =
        1 lbr$ini_control (help_index, func_read, type_help, help_namblk)
        IF (status) GOTO 7040
        GOTO 200
 7040   status = lbr$open (help_index, %descr ('sys$help:helplib.hlb'))
        IF (.NOT. status) GOTO 200
        status = lbr$get_help (help_index, 80, , ,
        1 key1 (1:key1len), key2 (1:key2len), key3 (1:key3len))
        IF (.NOT. status) CALL lib$signal (%VAL (status))
        status = lbr$close (help_index)
        IF (.NOT. status) CALL lib$signal (%VAL (status))
        GOTO 100
C
C       Error routines
C

C
C       No library name given
C
 8000   TYPE 9500
        GOTO 100
C
C       No library open
C
 8020   TYPE 9520
        GOTO 100
C
C       Format statements
C
 9000   FORMAT (' Action (0 for help): ',$)
 9020   FORMAT (' Commands:',/' 1 - Name library',/,
        1' 2 - Open or Create a text library',/,
        2' 3 - Replace/insert module',/,
        4' 4 - Extract module',/,
        3' 5 - List directory of SYS$HELP:HELPLIB.HLB',/,
        5' 6 - Lookup help text in SYS$HELP:HELPLIB.HLB')
 9040   FORMAT (' New library name: ',$)
 9050   FORMAT (A)
```

```
9060    FORMAT (' Maximum key length: ',$)
9080    FORMAT (' Module name: ',$)
9100    FORMAT (' Enter text. Terminate with a Control-Z:')
9110    FORMAT (Q, A)
9200    FORMAT (' Enter KEY1: ',$)
9220    FORMAT (' Enter KEY2: ',$)
9240    FORMAT (' Enter KEY3: ',$)
9400    FORMAT (' Module to extract: ',$)
9500    FORMAT (' No library name given')
9520    FORMAT (' No library open')
9540    FORMAT
        1 (' Open existing library (0) or create new library (1): ',$)
        END

        INTEGER FUNCTION list_module (keyname, keyrfa)

        IMPLICIT INTEGER (A-Z)

        CHARACTER *(*) keyname

        TYPE *,keyname
        list_module = .true.
        RETURN
        END
```

❶ LBRDEMO calls 11 of the Librarian routines. To call these routines, LBRDEMO uses a FORTRAN COMMON block to access symbols that are unavailable to FORTRAN. The symbols specify the functions available for initializing a library and the options available for creating a text library. (See ❼)

The COMMON block is initialized in the VAX-11 MACRO source module DEMOMAC; the symbols are accessed by linking the two programs together. The Librarian routines are accessed by specifying an options file that requests the linker to include the Librarian shareable image. To assemble and link the two programs, you can create a command procedure that contains the following commands:

```
$ FORTRAN /LIST LBRDEMO
$ MACRO  /LIST DEMOMAC
$ LINK /EXE=DEMO /MAP=DEMO /FULL LBRDEMO, DEMOMAC, SYS$INPUT/OPTIONS
!
! OPTIONS INPUT
!
SYS$LIBRARY:LBRSHR/SHARE                          !Include librarian
```

This command procedure produces the executable image DEMO.EXE.

❷ DEMOMAC contains a subroutine, NAM_INIT, that initializes an array to be used as a VAX-11 RMS NAM block when retrieving help messages from the system help library. (See ⓬ for information on how to use the NAM block option; for an example of opening a library without the NAM block option, see ❺)

❸ The subroutine RETRIEVE_RMSSTV returns the VAX-11 RMS STV value from the last library operation (see the VAX-11 Record Management Services Reference Manual). LBRDEMO does not call RETRIEVE_RMSSTV. However, if it were to be included, it would be declared as INTEGER*4, and the call would be ISTV=RETRIEVE_RMSSTV(0).

DEMOMAC also contains a subroutine, SET_LOCATE_MODE, that you can use to specify the VAX-11 RMS locate mode of record transfer. LBRDEMO does not call this subroutine. However, if this subroutine were to be included, it would be called after the LBR$OPEN routine (see the program comment following line 3070). For more information on using locate mode, see the VAX-11 Record Management Services Reference Manual.

❹ The main part of LBRDEMO, beginning at line 100, asks what library operation you want to perform. By typing "0" (for help), you request an operation menu that lists the program's six library functions:

1. Name library

2. Open or create a text library

3. Replace/insert module

4. Extract module

5. List directory of SYS$HELP:HELPLIB.HLB

6. Lookup help text in SYS$HELP:HELPLIB.HLB

Each operation on the menu references at least one of the Librarian routines. After the Librarian performs the requested operation, it returns you to this menu. At this point, you can either perform another operation by typing the appropriate number, or you can terminate the program by typing <CTRL/Z>. Typing <CTRL/Z> performs the following functions:

a. Calls the LBR$CLOSE routine (line 300) to close a previously opened text library

b. Executes the STOP instruction to terminate the program

❺ The first operation on the menu allows you to name the library you want to access. If a text library has been previously opened, LBRDEMO calls the LBR$CLOSE routine (line 2000) to close the library. LBRDEMO then prompts you for the library name (line 2050), appends the text library file type TLB to the name you specify, and assigns the string to the symbol LIBRARY_NAME. The variable HAVE_NAME is set to true, and LBRDEMO returns to the operations menu.

❻ The second operation on the menu opens or creates a library using the library name you specify in 1. LBRDEMO asks whether you are opening an existing library or creating a new one (line 3010). If you want to create a library, the variable FUNCTION will take the value stored in FUNC_CREATE; if you are opening a library, the variable FUNCTION will take the value stored in FUNC_UPDATE. In either case, LBRDEMO calls the LBR$INI_CONTROL routine to initialize the text library index.

The three arguments to the LBR$INI_CONTROL routine specify that the library index is named TEXT_INDEX, the library function to be performed is contained in the variable FUNCTION, and the library type is TEXT. The variable FUNCTION is accessed through the COMMON block. Note that the call to LBR$INI_CONTROL initializes a library index without using the NAM block option. Instead, the call to LBR$OPEN (line 3070) opens a library using the specifications contained in the LIBRARY_NAME argument.

Note also that the call to the LBR$INI_CONTROL routine does not enable you to access a library; it merely initializes a library

index to which the other Librarian routines can refer. The subsequent call to LBR$OPEN opens the library for access. If you try to perform a library operation before initializing a library index and opening a library file, LBRDEMO will print "No library name given" (line 4000) and will return you to the operations menu.

❼ If you are opening an existing library, the LBR$OPEN routine will open for access the library specified in LIBRARY_NAME. If you are creating a new library, the LBR$OPEN routine will use the information in the CREATE_OPTIONS array (line 3070) to create the library. The symbols in the create options array are accessed through the COMMON block. You must supply the maximum key length of the library (line 3050); the rest of the create options are predefined by the program.

❽ After you have initialized and opened (or created) a library, you can insert or replace a text module (the third operation on the menu). The method by which the Librarian inserts a module depends on whether the key to the module already exists in the library index.

In the sequence beginning at line 4000, the variable REPLACING is associated with the status of the LBR$LOOKUP_KEY routine. This variable is later used to test whether the module being inserted already exists in the library (see ❾).

LBRDEMO creates the new text module by prompting you to specify the module name and enter the text (lines 4020 and 4100, respectively). As you enter the text records, LBRDEMO calls the LBR$PUT_RECORD routine to write the text records to the module you specified. Note that the first call to LBR$PUT_RECORD fills in the MODULE_RFA argument with the address of the module. In this way, the first call to LBR$PUT_RECORD provides a mechanism for accessing the new module in subsequent calls to the Librarian. After the module is written, LBRDEMO calls the LBR$PUT_END routine (line 4200) to terminate the writing sequence.

❾ LBRDEMO inserts the module into the library using the LBR$REPLACE_KEY routine. If the key you are inserting does not already exist in the module name table, LBR$REPLACE_KEY will insert the key into the index. The new key will point to the inserted module.

If the key you are inserting already exists, LBR$REPLACE_KEY must perform a replace operation. In a replace operation, the Librarian changes the index key from the address specified in OLD_MODULE_RFA to the address specified by MODULE_RFA. The key then points to the new module, but the old module still exists physically in the library. Therefore, after the call to LBR$REPLACE_KEY, LBRDEMO must delete the old module.

If the value of REPLACING is true, the call to LBR$LOOKUP_KEY will find a module with the same name and return its address in the location specified by the argument OLD_MODULE_RFA. LBRDEMO then calls the LBR$DELETE_DATA routine (line 4300) to delete physically the old module from the library.

⑩ The fourth operation on the menu extracts a module from the currently open library and displays it on the terminal. LBRDEMO first prompts you for the module you want to extract, then calls LBR$LOOKUP_KEY. The call to LBR$LOOKUP_KEY searches TEXT_INDEX for the key specified by the MODULE_NAME argument. If the key is found, LBR$LOOKUP_KEY will fill in the array specified by MODULE_RFA with the record's file address (RFA) of the module you want to read.

Once the module is found, LBRDEMO calls the LBR$GET_RECORD routine to retrieve the text records. The first argument to the routine, STRING_DESC, specifies the input buffer that receives the text record. The second argument, DYN_STRING, is an optional argument that receives the actual length and address of the text record read.

The program sequence beginning at line 5100 constitutes a read loop. As LBR$GET_RECORD returns each text record, LBRDEMO checks for the end of file (RMSEOF). If it is not at the end of the file, LBRDEMO will call the Run-Time Library procedure LIB$PUT_OUTPUT to output the text record described by the string descriptor DYN_STRING.

After LBR$GET_RECORD retrieves all the text records in the module, LBRDEMO returns you to the operations menu.

⑪ The fifth operation on the menu, beginning at line 6000, lists the contents of the system help library, SYS$HELP:HELPLIB.HLB.

The LBR$INI_CONTROL routine initializes the library index, called HELP_INDEX. The second argument, FUNC_READ, indicates that the library is initialized for reading. Therefore, any attempt to modify the library in subsequent calls will result in an error. The symbol FUNC_READ is accessed through the COMMON block.

The call to LBR$OPEN opens system help library for read access. The second argument to the routine contains the file specification for the library.

The call to LBR$GET_INDEX retrieves the contents of the library index specified by the first argument, HELP_INDEX, which was returned by the preceding call to LBR$INI_CONTROL. The second argument tells the routine to get index number 1. The third argument, LIST_MODULE, is the name of the user-supplied routine (at the end of the program) that LBR$GET_INDEX calls to return the list of index entries.

The final call in this sequence, LBR$CLOSE, closes the index returned by the LBR$INI_CONTROL routine and deallocates all of the space for processing the library.

⑫ The sixth and last operation on the menu retrieves help text from the system help library, SYS$HELP:HELPLIB.HLB. After prompting you for the help keys (line 7000), LBRDEMO calls the LBR$INI_CONTROL routine to initialize an index for the help library.

Note that this call to LBR$INI_CONTROL uses the VAX-11 RMS NAM block option to initialize and open the help library. When you first select operation 6, the Librarian fills in the NAM block, HELP_NAMBLK, with the file identification of the system help library. Thereafter, when you select operation 6, the Librarian uses the HELP_NAMBLK argument containing the file identification to reopen the library. Using the NAM block method for opening a library is much faster because no directory accesses are required. Although the NAM block argument is optional in the

LBR$INI_CONTROL routine, you should use it whenever you repeatedly open and close a library.

The LBR$OPEN routine opens the system help library in preparation for the call to LBR$GET_HELP. LBR$GET_HELP retrieves the help text associated with the key path specified in the argument list. The number 80 before the list of help keys optionally requests that the text be displayed in 80-character lines.

After printing the help text, LBRDEMO calls the LBR$CLOSE routine to close the library. You then return to the operations menu where you can terminate LBRDEMO by typing <CTRL/Z> (see 4).

7.6 MESSAGES FOR LIBRARY COMMAND AND LIBRARIAN ROUTINES

This section lists messages issued by the LIBRARY command and Librarian routines, and provides an explanation of each message and suggestions for user action in response to error and severe-error messages. The messages appear in decreasing order of severity. Informational and success messages inform you that the Librarian has performed the specified request. Warning messages indicate that the command may have performed some, but not all of your request, and that you need to verify command or program output. Error messages indicate that the command or program output is incorrect, although the system may attempt to continue execution. Severe-error messages tell you that the error required the system to stop execution of the command or program.

7.6.1 Messages For LIBRARY Command

This section lists the information, success, warning, error, and severe-error messages for the LIBRARY command.

7.6.1.1 Informational Messages

CNVRTING, library file-spec is a copy of old library file-spec

Explanation: Any modification to a Version 1.0 library automatically converts it to the new library format.

7.6.1.2 Success Messages

NORMAL, success

INSERTED, module module-name inserted in library file-spec

DELETED, module module-name deleted from library file-spec

REPLACED, module module-name replaced in library file-spec

REMOVED, symbol symbol-name removed from library file-spec

EXTRACTED, module module-name extracted from library file-spec

7.6.1.3 Warning Messages

DIFTYP, expected library file-spec to be library type

Explanation: The referenced library is actually of a different type from that specified in the command string.

User Action: No action is necessary. Processing continues based on the actual library type.

CLOSEIN, error closing input file

Explanation: This is an error detected by VAX-11 RMS.

User Action: Reenter the command line after taking corrective action based on the accompanying message.

CLOSEOUT, error closing output file

Explanation: This is an error detected by VAX-11 RMS.

User Action: Reenter the command line after taking corrective action based on the accompanying message.

COMCOD, compilation error in module module-name file library file-spec

Explanation: The object module you are inserting contains a compilation error, a warning, or an invalid compilation code.

User Action: Recompile the module before reentering the command string.

ENDWRNGMAC, .ENDM does not end macro macro-name in library file-spec

Explanation: The macro name following the .ENDM statement does not match the name of the macro it should end.

User Action: Reformat the macro source file.

EXTRAENDM, extraneous .ENDM in library file-spec

Explanation: The specified library contains a .ENDM statement that does not terminate any macro.

User Action: Reformat the macro source file.

NOHLPTXT, no level 1 help text found in input file-spec

Explanation: The specified input file does not contain a properly formatted help file.

User Action: Reformat the help input file before reentering the command line.

NOMACFOUND, no .MACRO found in input file-spec

Explanation: The specified input file does not contain a properly formatted macro.

User Action: Reformat the macro source file.

NOMTCHENDM, no matching .ENDM for macro macro name in library file-spec

Explanation: The specified macro does not contain a matching .ENDM statement.

User Action: Reformat the macro source file.

NOMTCHENDR, number missing .ENDR for macro macro-name in library file-spec

Explanation: The Librarian found a matching .ENDM statement before finding the required .ENDR statement.

User Action: Reformat the macro source file.

NOMTCHFOU, no matches found for module-name

Explanation: The module specified with the /ONLY qualifier is not in the module name table.

User Action: Make sure that the specified module exists.

TOOMNYENDR, too many .ENDR in macro macro-name in library file-spec

Explanation: The specified macro contains a nonmatching .ENDR statement.

User Action: Reformat the macro source file.

7.6.1.4 Error Messages

DELKEYERR, error deleting module-name from library file-spec

Explanation: The module you want to delete does not appear in the module name table; or a VAX-11 RMS error occurred; or there was not enough virtual memory.

User Action: Correct the problem and reenter the command line.

DELDATERR, error deleting data from library file-spec

Explanation: An error occurred during an attempt to delete the text; or a VAX-11 RMS error occurred; or there was not enough virtual memory.

User Action: Compress the old library before deleting the specified text.

DUPGLOBAL, global symbol symbol name from file library file-spec already in library library file spec.

Explanation: An attempt was made to insert an object module containing a global symbol that was already in the global symbol table.

User Action: Replace the module rather than insert it; or remove the global symbol from the library before inserting the module.

DUPMOD, module module-name from file library file-spec already in library file-spec

Explanation: The module you are inserting into the library already exists in the module name table.

User Action: Change the name of the module, or replace the existing module with the new one.

FAOFAIL, system service failure

Explanation: This is an unexpected internal consistency check.

User Action: Submit a Software Performance Report.

GSDTYP, module module-name file library file-spec has an illegal GSD record record type

Explanation: The specified object module is invalidly formatted.

User Action: Recompile the object module before inserting it into the library.

ILLKEYLVL, illegal key level number key keyname in library file-spec expected key number

Explanation: The help module you want to insert into the library is not formatted properly.

User Action: Reformat the module before inserting it into the library.

INDEXERR, index error in library file-spec

Explanation: An error occurred during an attempt to search the index; or a VAX-11 RMS error occurred; or there was not enough virtual memory available.

User Action: Compress the library before reentering the command string.

INSERTERR, error inserting module name in library file-spec

Explanation: The LIBRARY command could not insert the specified modules into the library for one of the following reasons:

- The modules are not formatted properly
- The organization of the input file is incorrect
- A VAX-11 RMS error occurred
- There was not enough virtual memory

User Action: Correct the error and insert the file.

KEYNAMLNG, key key-name name length illegal in library file-spec

Explanation: The name of the module you are inserting exceeds the name length limit for the library.

User Action: Rename the module before inserting it into the library.

LOOKUPERR, error looking up module-name in library file-spec

Explanation: The Librarian could not find the requested module in the module name table; or a VAX-11 RMS error occurred; or there was not enough virtual memory available.

User Action: Reenter the command line specifying an existing module name.

MACNAMLNG, macro macro-name is too long in library file-spec

Explanation: The name of the macro you are inserting into the library exceeds the name length limit.

User Action: Either rename the macro or extend the library's name length limit.

MODNAMLNG, illegal module name module-name of number characters in library file-spec

Explanation: The name of the object module you are inserting into the library exceeds the name length limit.

User Action: Rename the module before inserting it into the library.

NESTLEVEL, nesting level exceeded in macro macro name file library file-spec

Explanation: The specified macro has exceeded the nesting limit of 63.

User Action: Change the macro before inserting it into the library.

NOEOM, module module-name is not terminated with EOM record in library file-spec

Explanation: The object module you want to insert into the library does not contain a legal EOM record.

User Action: Recompile the object module.

NOMODNAM, no module name found for input-file-spec

Explanation: An attempt was made to insert unnamed text modules into a text library.

User Action: Use the /MODULE qualifier to name the text modules.

NOTOBJLIB, not an object library

Explanation: An attempt was made to cross reference a library or to use the /REMOVE qualifier in a macro, help, or text library.

User Action: Reenter the command line, specifying an object library that contains the requested symbol.

OPENIN, input file open error

Explanation: The LIBRARY command could not open the input file for one of the following reasons:

- The user directory or file is protected against read access
- A physical device problem; for example, the volume is not mounted
- The specified directory does not exist
- The specified file does not exist
- On create and compress operations, there is not enough space to allocate the new library file.

User Action: Correct the problem and reenter the command line.

READERR, error reading file file

Explanation: This is an error detected by VAX-11 RMS.

User Action: Check that the file exists and that you have read privilege to it.

RECLNG, illegal record length number in module module-name in library file-spec

Explanation: The specified module contains records exceeding the maximum record length of 2048 bytes.

User Action: Correct the record length before inserting the module in the library.

RECTYP, illegal record type type in module module-name in library file-spec

Explanation: The specified object module contains an illegal record type.

User Action: Recompile the module before reentering the command string.

SEQNCE, illegal record sequence in module module-name in library file-spec

Explanation: The object module you want to insert is illegally formatted.

User Action: Recompile the object module before inserting it into the library.

SPNAMLNG, PSECT module module-name file library file-spec has illegal length number

Explanation: The program-section name length exceeds the maximum length of 31 characters.

User Action: Recompile the module before reentering the command line.

STRLVL, object structure level number unsupported in module module-name in library file-spec

Explanation: The object file you are inserting is invalidly formatted.

User Action: Recompile the object file before inserting it into the library.

SYMNAMLNG, module type module module-name file library file-spec has illegal length number

Explanation: The specified module name exceeds the name length limit.

User Action: Make sure that all the symbol names in the module are less than the library's name length limit; or compress the library with a larger key length.

WRITEERR, error writing file

Explanation: This is an error detected by VAX-11 RMS.

User Action: Make sure that the file is open and that you have write privilege to it.

7.6.1.5 Severe Error Messages

BADKEY, illegal key

Explanation: The specified module is not valid.

User Action: Reenter the command line, specifying a valid module.

MHDERR, module header error for module name in library-file-spec

Explanation: The specified module has an invalidly formatted header.

User Action: Compress the library before reentering the command line.

INITERR, error initializing library-file-spec.

Explanation: There is not enough virtual memory available.

User Action: Increase your working set limit.

OPENOUT, error opening output file

Explanation: This is an error detected by VAX-11 RMS.

User Action: Make sure that the file is open and that you have write privilege to it.

7.6.2 Librarian Routines Messages

This section lists the success, warning, and error messages for the Librarian routines. Note that any of the Librarian routines may produce VAX-11 RMS errors. On a VAX-11 RMS I/O error, the variable LBR$GL_RMSSTV is the VAX-11 RMS STV value of the failing operation. On a successful call to LBR$OPEN, the VAX-11 RMS STV (Status Value) is the type of library opened. For more detailed information on VAX-11 RMS errors, see the VAX-11 Record Management Services Reference Manual.

7.6.2.1 Success Messages

NORMAL success

OLDLIBRARY old format library opened

7.6.2.2 Warning Messages

LBR$_HDRTRUNC, header truncated

Explanation: The buffer supplied to LBR$SET_MODULE is smaller than the header information.

User Action: Supply a larger buffer to LBR$SET_MODULE.

LBR$_LIBOPN, library already open

Explanation: The library you attempted to open is already open.

User Action: No action is necessary; processing continues.

LBR$_NOMATCHFOU, no match found

Explanation: The specified module does not appear in the module name table.

User Action: Make sure that the module exists in the current library.

LBR$_NULIDX, index is empty

Explanation: On a call to LBR$GET_INDEX, the specified index is empty.

User Action: No action is necessary; processing continues.

LBR$_OLDMISMCH, old format library type mismatch

Explanation: The requested Version 1.0 library is of a different type from that expected.

User Action: No action is necessary; processing continues based on the actual library type.

LBR$_RECTRUNC, record truncated

Explanation: The buffer supplied in the routine is too small for the record.

User Action: Supply a sufficiently large buffer to contain the record.

LBR$_STILLKEYS, keys still point at data

Explanation: Keys in other indexes still point to the text. Therefore, the call to LBR$DELETE_DATA did not delete the text.

User Action: No action is necessary; processing continues.

LBR$_TYPMISMCH, library type mismatch

Explanation: On a call to LBR$OPEN, the library type you requested conflicts with the library type you specified in the create-options argument.

User Action: No action is necessary; processing continues based on the actual library type.

7.6.2.3 Error Messages

LBR$_DUPKEY, duplicate key in index

Explanation: On a call to LBR$INSERT_KEY, the specified key already exists.

User Action: Call LBR$REPLACE_KEY to replace the existing key.

LBR$_ILLCTL, illegal control index

Explanation: Either the library is not open, or you have specified an invalid library index.

User Action: Make sure the library is open, and the library index is correct.

LBR$_ILLCREOPT, illegal create options

Explanation: The create options you specified for LBR$OPEN are invalid; or no options were specified.

User Action: Correct the create options before calling the routine.

LBR$_ILLIDXNUM, illegal index number

Explanation: The index number you specified is invalid.

User Action: Correct the index number before calling the routine.

LBR$_ILLFMT, illegal library format

Explanation: The file you specified calling LBR$OPEN is not an actual library.

User Action: Specify an actual library.

LBR$_ILLFUNC, illegal library function

Explanation: The library function you specified in LBR$OPEN is invalid.

User Action: Correct the library function before calling the routine.

LBR$_ILLOP, illegal operation for access requested

Explanation: You have attempted to modify a Version 1.0 library; or you have attempted to modify a library that you opened for read access.

User Action: Compress the library into the current format; or change the access privileges to the library file.

LBR$_ILLTYP, illegal library type

Explanation: The type of library specified in LBR$INI_CONTROL is invalid.

User Action: Correct the library type before calling the routine.

LBR$_INVKEY, invalid key

Explanation: The specified key is either of 0 length or it is greater than the maximum allowable length.

User Action: Either change the module name, or change the library's name length limit by compressing the library.

LBR$_INVNAM, invalid NAM block

Explanation: The NAM block passed to LBR$INI_CONTROL is invalid.

User Action: Correct the NAM block before calling the routine.

LBR$_INVRFA, invalid RFA

Explanation: The specified record's file address (RFA) is invalid.

User Action: Correct the RFA before calling the routine.

LBR$_KEYNOTFND, key not found

Explanation: The Librarian could not find the key you specified in the key-name argument to the LBR$LOOKUP_KEY routine.

User Action: Correct the key-name argument before calling the routine.

LBR$_LIBNOTOPN, library not open

Explanation: Except for LBR$INI_CONTROL and LBR$OPEN, all routines require that the library be open.

User Action: Open the library before calling the routine.

LBR$_LKPNOTDON, lookup has not been done

Explanation: Before calling LBR$GET_RECORD, you must first call LBR$LOOKUP_KEY.

User Action: Call LBR$LOOKUP_KEY before calling LBR$GET_RECORD.

LBR$_NOFILENAM, no file specification found

Explanation: In the LBR$OPEN routine, either the fns argument was not supplied, or the NAM block was not filled in.

User Action: Correct the problem before calling the routine.

LBR$_NOHLPTXT, help text not found

Explanation: There is no help text associated with the specified key path.

User Action: Check that the help module is properly formatted.

LBR$_NOTHLPLIB, not a HELP library

Explanation: The library type specified in LBR$GET_HELP is not a help library.

User Action: Make sure that the library is a help library before calling the routine.

LBR$_RECLNG, illegal record length

Explanation: The record length exceeds 2048 bytes.

User Action: Reformat the module.

LBR$_RFAPASTEOF, VBN in map block request past EOF

Explanation: This is an internal consistency check.

User Action: Submit a Software Performance Report.

LBR$_TOOMNYLIB, too many libraries open

Explanation: Only 16 libraries can be open concurrently.

User Action: Use LBR$CLOSE to close any library you do not need to access.

LBR$_UPDURTRAV, index update during traverse illegal

Explanation: The routine that you specify in the routine-name argument cannot contain any calls to either LBR$DELETE or LBR$INSERT_KEY.

User Action: Remove any embedded calls to either LBR$DELETE or LBR$INSERT_KEY before calling the routine.

CHAPTER 8

MESSAGE UTILITY

The VAX-11 Message Utility allows programmers to control the generation of messages by VAX/VMS.

Messages are produced by VAX-11 software products under many circumstances -- for example, when a routine has run successfully, when an error has occurred, or when a default value has been assigned.

Messages are displayed to the user as a line of alphanumeric codes and text that explains the condition that caused the message to be issued. A message is represented to the VAX-11 processor as a longword called the message code. This 32-bit value can be referred to in programs by means of a global symbol called the message symbol.

The information that appears in the message that users receive, the values that make up the message code, and the characters that make up the message symbol are all defined in files called message source files. VAX/VMS has a file of system message information called SYSMSG.EXE.

You can use your own message files by means of the VAX-11 Message Utility, following these steps:

- Use a text editor to create a source file that specifies the information used in messages, message codes, and message symbols.
- Use the MESSAGE command to compile this source file.
- Link the resulting object module, either by itself or with another object module containing a program.
- Run your program so that the messages are accessed, either directly or through the use of pointers.

You can modify messages at run time by using the SET MESSAGE command or by using pointers to message information. These features allow you to suit messages to the requirements of your installation.

This chapter begins with a description of messages, the message code, and the message symbol. It then explains the components of a message source file, and how the file can be compiled and linked. It describes different methods of changing messages at run time, and concludes with a list of the messages issued by the Message utility itself.

8.1 THE FORMAT OF MESSAGES

Messages have the following format:

%FACILITY-L-IDENT, message-text

% and ,

The percent sign (%) and comma (,) are included as delimiters if any of the first three fields -- FACILITY, L, or IDENT -- is present.

FACILITY

The abbreviated name of the software product that issued the message. You specify the facility name to be used in your message source file by means of the facility definition, described in Section 8.3.1.1. The FACILITY field can contain up to nine characters from the character set described in Section 8.3.1.

You can suppress the appearance of FACILITY for your process by means of the /NOFACILITY qualifier on the DCL command SET MESSAGE, described in Section 8.4.2.

L

An indicator showing the severity level of the condition that caused the message. There are five levels, represented by the following codes:

Code	Function
S	Success
I	Informational
W	Warning
E	Error
F	Fatal or severe

You set the severity level of messages in your message source file using either the severity definition, described in Section 8.3.1.2, or a severity qualifier on the message definition, described in Section 8.3.1.4.

You can suppress the appearance of the severity level indicator for your process by means of the /NOSEVERITY qualifier on the DCL command SET MESSAGE, described in Section 8.4.2.

IDENT

A symbol of up to nine characters representing the message. Valid characters are described in Section 8.3.1. You specify the symbol in the message definition line of your message source file, described in Section 8.3.1.4.

You can suppress the appearance of the IDENT symbol for your process by means of the /NOIDENTIFICATION qualifier on the DCL command SET MESSAGE, described in Section 8.4.2.

message-text

A brief explanation of the cause of the message. You specify the message text in the message definition line of your message source file, described in Section 8.3.1.4.

If you suppress FACILITY, L, and IDENT, the first character of the message text will be capitalized by the Put Message ($PUTMSG) system service.

You can suppress the appearance of the message text for your process by specifying the /NOTEXT qualifier on the DCL command SET MESSAGE, described in Section 8.4.2.

The message can also include up to 255 formatted-ASCII-output (FAO) arguments; that is, character strings that can be used to display, for example, the instruction at which an error occurred or a value of which the user should be aware. The following sample message includes the file specification as an FAO argument:

```
%TYPE-W-OPENIN, error opening _DB0:[MARCEL]BBBB.FOR; as input
```

8.2 THE MESSAGE CODE AND THE MESSAGE SYMBOL

Messages are formatted by the Put Message ($PUTMSG) system service, described in the VAX/VMS System Services Reference Manual. The system service finds the information to use in the message by using a message argument vector. The message argument vector includes a 32-bit value that uniquely identifies the message. This 32-bit value is called the message code.

The message code is made up of the following elements, which are described individually in Section 8.3.1:

- The severity level defined in the severity definition or message definition
- The message number assigned automatically by a message definition or specified with the message number specifier
- The facility number defined in the facility definition
- The customer facility bit of the control area, the setting of which can be inhibited in the facility definition

Figure 8-1 shows the arrangement of the bits in the message code. The message code is described fully in the VAX-11 Architecture Handbook description of "Condition Value."

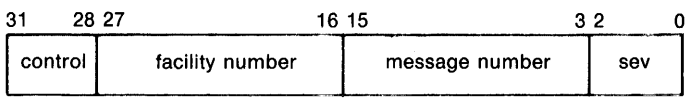

Figure 8-1 Message Code

The message symbol is the symbol that represents the message code. It appears in the object module (the compiled message file) as a global symbol.

The message symbol is constructed of the following elements, described in Section 8.3:

- The symbol prefix defined in the facility definition
- The symbol name defined in the message definition

8.3 CONSTRUCTING MESSAGES

You construct messages by creating a message source file that contains the information that you want to include in the message, the message code, and the message symbol. You then compile the message source file with the Message compiler and link the resulting object module with the VAX-11 Linker. This section describes the contents of the message source file and the use of the compiler and linker to convert the message source file into usable form. It concludes with a sample program that generates messages at run time.

8.3.1 The Message Source File

The message source file contains the information that makes up the message, the message code and the message symbol. The file is made up of statements that establish the various fields of the message, define symbols, and control the output listing of the file. The message source file has the default file type MSG.

The elements of the message source file, described in the following sections, are:

- Facility definition
- Severity definition
- Message number specifier
- Message definition
- Literal directive
- Listing directives
- End statement

The format of each statement in the message source file is described in the appropriate section below. A statement in a message source file can take up any number of lines; text that reaches the end of a line and is to be continued on the next line must end with a hyphen (-). The only exceptions to this are the listing title specified with the .TITLE directive and the message text specified in the message definition, which must occupy only one line.

Any line in the message source file can include a comment, delimited by an exclamation point (!). In any line, you can freely insert extra spaces and tabs to improve readability.

Symbols defined in the Message utility can include any of the following characters:

A - Z
a - z
1 - 9
$ (dollar sign)
_ (underline)

Expressions used in the Message utility can include any of the following radix operators to specify the radix of a numeric value:

Operator	Radix	Example
^X	Hexadecimal	^X10
^O	Octal	^O30
^D	Decimal	^D16

The default radix is decimal.

Expressions can include symbols and the unary operators plus sign (+), which assigns a positive value, and minus sign (-), which assigns a negative value. Expressions can also include the following binary operators:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
@	Arithmetic shift

Expressions can include parentheses as special operators. Expressions enclosed in parentheses are evaluated first. Nested parenthetical expressions are evaluated inside to outside.

8.3.1.1 **The Facility Definition** -- The texts of messages are grouped in a message source file by the facility (software product) to which they apply, broken down by severity levels. Therefore, a facility definition, specifying the facility to which messages will apply, is the first definition in a message source file. All of the lines following a facility definition apply to that facility, until an end statement or another facility statement is reached.

The facility definition has the format:

.FACILITY[/qualifier,...] facnam[,]facnum [/qualifier,...]

qualifier

One of the qualifiers listed in Table 8-1.

facnam

The facility name, which will be used as the facility field of the message and in the symbol that represents the facility number. It can have up to nine characters. Facility names for VAX/VMS facilities are listed in the VAX/VMS System Messages and Recovery Procedures Manual.

facnum

The facility number, a decimal value in the range of 1 to 32768, or an expression that evaluates to a value in that range. (For information on expressions, see the VAX/VMS Command Language User's Guide.) Facility numbers are usually assigned by the system manager so that no two facilities have the same number. The facility number is used to construct the 32-bit value of the message code.

Note that both the facility name and the facility number are required. They can be separated by a comma or by any number of spaces or tabs.

The facility definition creates a global symbol of the form:

facnam$_FACILITY

This symbol can be used to refer to the facility number assigned to the facility.

Table 8-1
Facility Definition Qualifiers

Qualifier	Function
/PREFIX=prefix	Defines an alternate symbol prefix to be used in the message symbol for all messages referring to this facility. The alternate prefix can have up to nine characters. The default symbol prefix is the facility name followed by an underline (_). If /SYSTEM is also specified, the default prefix is the facility name followed by a dollar sign and an underline ($_).
/SHARED	Inhibits setting the facility specific bit in the message codes. This qualifier is used only for system service and shared messages. This qualifier is reserved for DIGITAL use.
/SYSTEM	Inhibits setting the customer facility bit in the message codes. This qualifier is reserved for DIGITAL use.

8.3.1.2 **The Severity Definition** -- Following the facility definition, the message source file generally contains a severity definition specifying the severity level to be associated with the messages that follow. You must include a severity definition if you do not specify the severity individually on each message definition (see Section 8.3.1.4).

The severity definition has the format:

```
                 ( SUCCESS       )
                 ( INFORMATIONAL )
                 ( WARNING       )
.SEVERITY        ( ERROR         )
                 ( SEVERE        )
                 ( FATAL         )
```

SEVERE is equivalent to FATAL and can be used interchangeably with it; the severity level code for both of these, as described in Section 8.1, is F.

If you attempt to define a message without specifying a severity level, an error will result. A new facility definition cancels the severity level in effect before it.

8.3.1.3 **The Message Number Specifier** -- The message number is a value used in constructing the message code that represents the message (see Section 8.2). All of the messages following a facility definition are numbered sequentially, beginning with 1 after each facility definition.

In some cases, you may need to supersede this numbering system -- for example, if you want to reserve some message numbers for future assignment. You can specify a message number of your choice using the message number specifier, .BASE, which has the following format:

```
.BASE     number
```

number

A message number to be associated with the next message definition, or an expression that is evaluated as the desired number. This message number is used as a base for the sequential numbering of all messages that follow until another .BASE is encountered or until the end of the messages belonging to the facility.

8.3.1.4 **The Message Definition** -- The message definition specifies the body of the message symbol, the message text, and the number of arguments that can be printed with the message. Any number of message definitions can follow the severity definition. The message definition has the format:

```
name[/qualifier,...]    <message-text>[/qualifier,...]
```

name

Up to nine characters. This symbol name is combined with the symbol prefix defined in the facility definition to make up the message symbol.

The symbol name is used in the IDENT field of the message (see Section 8.1) unless the /IDENTIFICATION=name qualifier is specified in the message definition, as described in Table 8-2.

/qualifier

Any of the qualifiers listed in Table 8-2. Qualifiers can be placed before or after the message text, in any order.

message-text

An explanation of the condition that caused the message to be issued. The message text can be delimited either by angle brackets (<>), as shown above, or by quotation marks ("). The text can be up to 255 bytes long; however, you cannot continue the delimited text onto another line. The message text can include directives that insert ASCII strings into the resulting message; these directives are used by the Formatted ASCII Output ($FAO) system service and are described in the VAX/VMS System Services Reference Manual. If you include an FAO directive, you must also use the /FAO_COUNT qualifier, described in Table 8-2.

Table 8-2
Message Definition Qualifiers

Qualifier	Function
/FAO_COUNT=n	Specifies the number of FAO arguments to be included in the message at execution time. (See the VAX/VMS System Services Reference Manual for an explanation of FAO arguments.) The value n must be a decimal number in the range 0 through 255. The $PUTMSG system service uses n to determine how many arguments are to be given to the $FAO system service when constructing the final message text. The default value for n is zero.
/IDENTIFICATION=name	Specifies an alternate character string to be used as the IDENT field of the message (see Section 8.1). The string can include up to nine characters. If this qualifier is not specified, the symbol name defined in the message definition (see above) will be used in the IDENT field of the message.
/USER_VALUE=n	Specifies an optional user value that can be associated with the message. The value n must be a decimal number in the range of 0 through 255. The default is zero. The value can be retrieved by the Get Message ($GETMSG) system service for use in classifying messages by type or by action to be taken.

(continued on next page)

Table 8-2 (Cont.)
Message Definition Qualifiers

Qualifier	Function
/SUCCESS /INFORMATIONAL /WARNING /ERROR /SEVERE /FATAL	Specify the severity level to be associated with the message. You can use these qualifiers to supersede the severity level defined in a severity definition. You can also use these qualifiers instead of including severity definitions in your message source file. Only one severity qualifier can be included per message definition.

8.3.1.5 **The Literal Directive** -- The literal directive allows you to define global symbols in your message source file. You can either assign values to these symbols or use the default values provided by the statement. The .LITERAL directive has the form:

```
.LITERAL      symbol[=value][,...]
```

symbol

A symbol name.

value

Any valid expression. If value is omitted, a default value is assigned. The default value is 1 for the first symbol in the statement; for subsequent symbols in the same statement, the default value is 1 plus the last value assigned.

You can assign default values to a list of symbols. For example:

```
.LITERAL    A,B,C
```

The values of A,B, and C will be 1, 2, and 3.

You can use the .LITERAL directive to define a symbol as the value of another previously defined symbol, or as an expression that results from operations performed on previously defined symbols. In the following example, symbols defined in the facility and message definitions are used to assign values to symbols created with the .LITERAL directive.

```
.FACILITY       SAMPLE,1/PREFIX=MSG$_
.SEVERITY       ERROR
FIRST           <first error>

.
.
.

LAST            <last error>
.LITERAL        LASTM$G=MSG$_LAST

.LITERAL        NUMSG=(MSG$_LAST@-3)-(MSG$_FIRST@-3)    ! # of messages
```

The first .LITERAL directive defines a symbol that has the value of the last 32-bit message code defined. The second .LITERAL directive defines the total number of messages in the source file.

8.3.1.6 **Listing Directives** -- You can use two special statements to control the output listing that is produced when you compile your message source file. These statements are the .PAGE directive and the .TITLE directive.

The .PAGE directive enables you to force page breaks in the output listing. It has the format:

```
.PAGE
```

You can only specify one page break with any one .PAGE directive; however, you can use the .PAGE directive as often as you like.

The .TITLE directive enables you to specify the module name and title text that will appear on the top of each page of the listing file. It has the format:

```
.TITLE modname [listing-title]
```

modname

A character string of up to 31 characters that will appear in the object module as the module name.

listing-title

Text to be used as the title of the listing. The text begins with the first nonblank character after the module name through the end of the line. The listing title cannot be continued onto another line.

8.3.1.7 **The End Statement** -- A group of message definitions is terminated by either: another severity definition, to begin a new group of another severity; an end statement, which terminates the entire list of messages for the facility; or a new facility statement. The end statement has the format:

```
.END
```

A new facility statement performs an implicit .END.

8.3.1.8 **Sample Message Source File** -- The following sample message source file illustrates the various elements described above.

```
.TITLE          SAMPLE Error and Warning Messages
.FACILITY       SAMPLE,1/PREFIX=ABC_
.SEVERITY       ERROR

UNRECOG         <Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG           <Ambiguous keyword>

.SEVERITY       WARNING
.BASE           10
SYNTAX          <Invalid syntax in keyword>

.END
```

The messages defined in this message source file belong to a facility with the name SAMPLE and the facility number 1. The first two messages have the severity level E; the third message has the severity level W.

The first message definition above includes the FAO directive !AS (which inserts an ASCII string at the end of the message text) and the corresponding qualifier /FAO_COUNT, as described in Section 8.3.1.4.

The message symbols defined in this message source file are ABC_UNRECOG, ABC_AMBIG, and ABC_SYNTAX. The message numbers are 1, 2, and 10.

8.3.2 Compiling the Message Source File

Message source files must be compiled into object modules before the messages defined in them can be used. You compile your message source file by issuing the MESSAGE command in response to the DIGITAL Command Language (DCL) prompt. The MESSAGE command can also be used to create object modules that do not contain message data; instead, they contain pointers to files that contain message data. These pointers are described in Section 8.4.1.

The MESSAGE command has the following format:

```
MESSAGE[/qualifier,...] file-spec[,...]
```

/qualifier

Any of the qualifiers listed in Table 8-3. Note that some qualifiers are mutually exclusive.

file-spec

The message source file to be compiled. If you do not specify a file type, the default is MSG.

You can specify more than one message source file, separated by either commas (,) or plus signs (+). The files will be concatenated and compiled as a single file.

If you specify SYS$INPUT, the message source file(s) must immediately follow the MESSAGE command in the input stream, and both the object module name (given by the /OBJECT qualifier) and the listing file name (given by the /LIST qualifier) must be explicitly stated.

For your convenience, you can put message object modules into object module libraries. These libraries can then be linked with facility object modules.

Table 8-3
MESSAGE Command Qualifiers

Qualifier	Function
/FILE_NAME=file-spec /NOFILE_NAME	Specifies whether the object module contains a pointer to a file containing messages. (Pointers are described in Section 8.4.1.) The default is /NOFILE_NAME, indicating that the object module contains only compiled message information, and no pointers. Whenever you specify /FILE_NAME=file-spec, the /NOTEXT qualifier is implied; that is, the /FILENAME and /TEXT qualifiers are mutually exclusive. The /OBJECT qualifier must be in effect, either explicitly or implicitly. The file specification identifies a nonexecutable message file, as explained in Section 8.4.1. The default device and directory for the file specification is SYS$MESSAGE; and the default file type is EXE. No wild card characters are allowed in the file specification.
/LIST[=file-spec] /NOLIST	Controls whether an output listing is created, and optionally provides an output file specification for the listing. When you compile message source files in batch mode, the output listing is created by default. However, in interactive mode, the default is to produce no output listing. The default file type for listing files is LIS. The default device and directory are your default device and directory. No wild card characters are allowed in the file specification.
/OBJECT[=file-spec] /NOOBJECT	Controls whether an object module is created by the message compiler, and optionally provides a file specification for the object module. By default, the compiler creates an object module with the same name as the first message source file and with the file type OBJ. The default device and directory are your default device and directory. No wild card characters are allowed in the file specification.

(continued on next page)

Table 8-3 (Cont.)
MESSAGE Command Qualifiers

Qualifier	Function
/SYMBOLS /NOSYMBOLS	Controls whether global symbols will be present in the object module. By default, object modules are created with global symbols. The /SYMBOLS qualifier requires that the /OBJECT qualifier be in effect, either explicitly or implicitly.
/TEXT /NOTEXT	Controls whether the data portion of the object module, containing the information specified in facility, severity, and message definitions, is present in the object module. (Section 8.4.1 describes the use of pointers, which require that data not be present in the object module.) The default is /TEXT. The /TEXT and /FILE_NAME qualifiers are mutually exclusive. The /TEXT qualifier requires that the /OBJECT qualifier be in effect, either explicitly or implicitly. The /NOTEXT qualifier can be used with the /SYMBOLS qualifier to produce an object module containing only global symbols.

8.3.3 Linking the Message Object Module

Before your messages can be used, your program must be linked by the VAX-11 Linker. The VAX-11 Linker resolves symbolic and library references and assigns virtual memory addresses to the relative addresses assigned by the compiler. It produces an executable image file with the file type EXE, which can be run on a VAX-11 processor.

The message object module that results from compiling your message source file can be linked in two different ways. It can be linked with an object module from the facility to which it applies, creating one executable image that contains both the facility code and the message data. Or, it can be linked by itself to create a nonexecutable message file. A nonexecutable message file can be used as a process permanent message file (see Section 8.4.2) or can be referenced at run time by a pointer (see Section 8.4.1).

Figure 8-2 shows the two ways of linking a message object module.

To link your message object module, issue the DCL command LINK in response to the DCL prompt, as described in the VAX-11 Linker Reference Manual. The following is the command that links the object module COBOLCODE.OBJ and the message object file COBOLMSG.OBJ. The command creates an image map file. The resulting executable image file is named COBOLCODE.EXE.

```
LINK/MAP    COBOLCODE,COBOLMSG
```

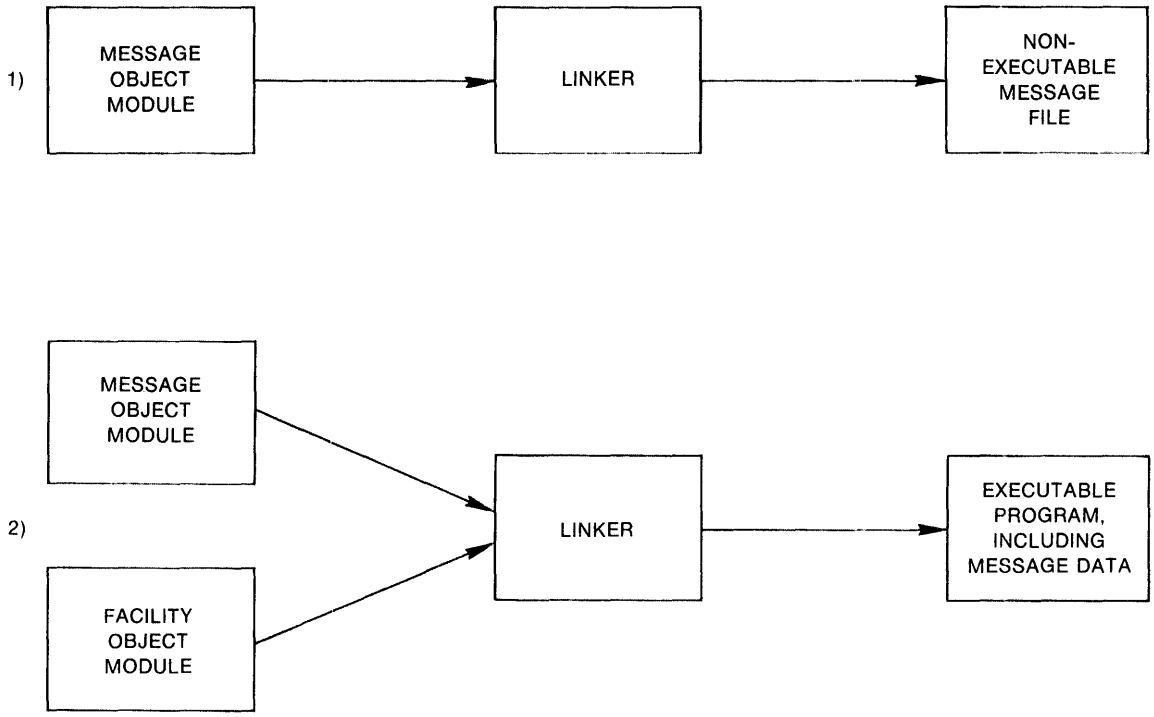

Figure 8-2 Linking a Message Object Module

8.3.4 Running a Program with Messages

This section shows how a program, linked with a message object module, produces messages when run.

The program is a FORTRAN program named TEST.FOR. It contains the following lines:

```
EXTERNAL MSG_SYNTAX,MSG_ERRORS
CALL LIB$SIGNAL(MSG_SYNTAX,%VAL(1),'ABC')
CALL LIB$SIGNAL(MSG_ERRORS)
END
```

This program calls the run-time procedure LIB$SIGNAL, described in the VAX-11 Run-Time Library Reference Manual. The message symbols MSG_SYNTAX and MSG_ERRORS are included as arguments in the procedure calls. The function %VAL is a required FORTRAN compile-time function. The first call also includes the string 'ABC' as an FAO argument.

You compile the FORTRAN program by issuing the following command:

```
$ FORTRAN TEST
```

This command results in an object module named TEST.OBJ.

The message source file, TESTMSG.MSG, contains the following lines:

```
.FACILITY       EFGH,1 /PREFIX=MSG_
.SEVERITY       ERROR
SYNTAX          <Syntax error in string '!AS'>/FAO=1
ERRORS          <Errors encountered during processing>
.END
```

You compile the message source file by issuing the following command:

```
$ MESSAGE TESTMSG
```

This command results in a message object module named TESTMSG.OBJ.

You link the two object modules by issuing the following command:

```
LINK/NOTRACE TEST+TESTMSG
```

This command results in an executable program named TEST.EXE. You run this program by issuing the following command:

```
RUN TEST
```

The following messages are issued when the program is run:

```
%EFGH-E-SYNTAX, Syntax error in string 'ABC'
%EFGH-E-ERRORS, Errors encountered during processing
```

8.4 CHANGING MESSAGES

Under some circumstances, you may want to change the messages for a facility that runs on your VAX-11 processor. You can make run-time changes on two levels:

1. per image, by using pointers to message data
2. per process, by using the DCL command SET MESSAGE

Using pointers is described in Section 8.4.1; using the SET MESSAGE command is described in Section 8.4.2.

8.4.1 Pointers to Message Data

If you have linked your message object module directly with the facility object module, you will have to alter the resulting executable image file to change the message data included in it. This can be time consuming and the resources needed to link the image may not be available. To avoid having to alter the executable image, you can use pointers to a message file instead of linking the message data into the image.

A pointer is created by referring to a non-executable message image in a MESSAGE command, using the /FILE_NAME qualifier (described in Section 8.4). The non-executable message file is a message source file that has been compiled and linked by itself.

The MESSAGE/FILE_NAME command results in an object module containing only global symbols and the file specification of the message file, which can then be linked with facility object modules.

An object module containing a pointer to message files should have a different file name from the module that actually contains message data.

The following command creates an object module named MESPNTR.OBJ, which contains a pointer to the non-executable message file COBOLMF.EXE. (COBOLMF.EXE was created by compiling the message file COBOLMSG.MSG and linking the resulting object module by itself with the qualifier /EXECUTABLE=COBOLMF.) Note that it is not necessary to include the file type EXE in the /FILE_NAME qualifier, because EXE is the default. The object module, MESPNTR.OBJ, contains the global symbols defined in the message source file COBOLMSG.MSG.

```
MESSAGE/FILE_NAME=COBOLMF /OBJECT=MESPNTR COBOLMSG
```

When the resulting facility image file is run, the message data is retrieved from the message file COBOLMF by the $GETMSG system service. Figure 8-3 illustrates the relationship of the files affected by this command.

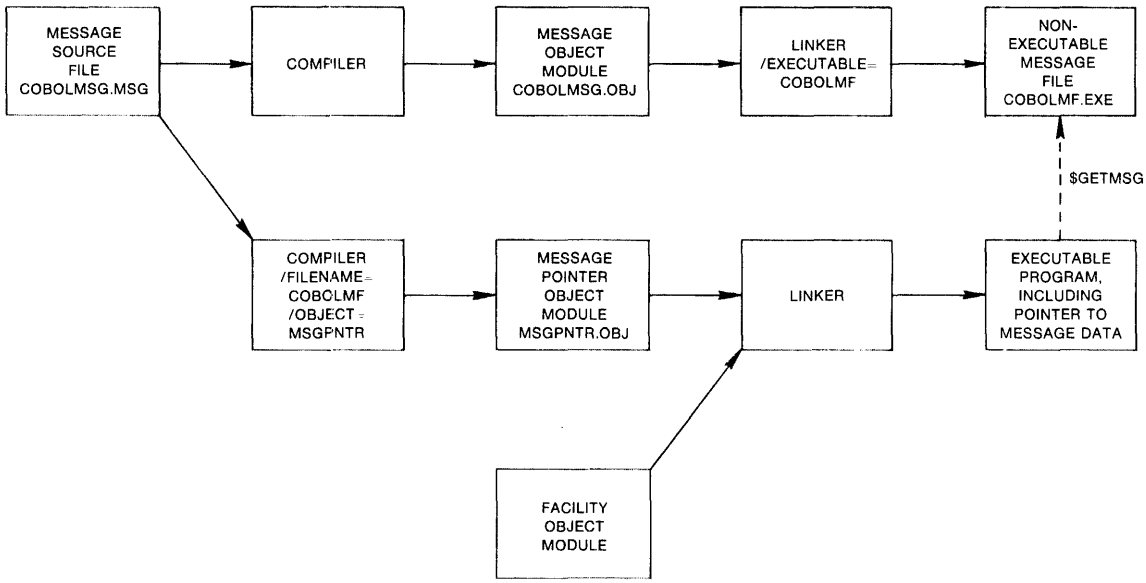

Figure 8-3 Creating a Message Pointer

8.4.2 The SET MESSAGE Command

You can override or supplement the system messages on your system by using the DCL command SET MESSAGE. This command allows you to suppress for your process the various fields of the messages (described in Section 8.1) or to substitute the message data in a nonexecutable message image for the system message data.

The SET MESSAGE command has the following format:

```
SET MESSAGE[/qualifier...]    [file-spec]
```

/qualifier

A qualifier listed in Table 8-4. Multiple command qualifiers can be used, specified in any order.

file-spec

An optional nonexecutable message file. The default file type is EXE; no wild card characters are allowed in the file specification.

The specified message file supersedes any per-process message file already in effect; only one per-process message file can be in effect at any time.

The messages contained in the specified message file are searched by the $GETMSG system service after the per-image messages and before the system-wide messages.

Table 8-4
SET MESSAGE Qualifiers

Qualifier	Function
/DELETE	Removes the process message file from your process. This qualifier cannot be used if you have included a file specification in the SET MESSAGE command; selecting a new per-process message file automatically removes any existing process message file.
/FACILITY /NOFACILITY	Control whether the facility name field (see Section 8.1) is displayed for all messages that occur in your process.
/IDENTIFICATION /NOIDENTIFICATION	Control whether the IDENT field (see Section 8.1) is included for all messages that occur in your process.
/SEVERITY /NOSEVERITY	Control whether the severity level indicator (see Section 8.1) is displayed for all messages that occur in your process.
/TEXT /NOTEXT	Control whether the message text field is displayed for all messages that occur in your process.

Examples

1. $ SET MESSAGE MYMSG

   The SET MESSAGE command specifies that the message information in MYMSG.EXE supplements the existing system messages.

2. $ TYPE BBBB.FOR
   %TYPE-W-OPENIN, error opening DB1:[MARCEL]BBBB.FOR; as input
   -RMS-E-FNF, file not found

   $ SET MESSAGE/NOIDENTIFICATION

   $ TYPE BBBB.FOR
   %TYPE-W, error opening DB1:[MARCEL]BBBB.FOR; as input
   -RMS-E, file not found

When the first TYPE command is entered, error messages contain all fields. Entering the SET MESSAGE command with the /NOIDENTIFICATION qualifier eliminates the IDENT field from the messages that are subsequently issued.

8.5 MESSAGE UTILITY MESSAGES

This section lists, in alphabetical order, the error messages issued by the Message compiler, with an explanation of the condition that caused the error and a recommended user response to the message. Most of the user responses entail changing the message source file and re-entering the MESSAGE command.

All of these messages have the severity level ERROR.

MESSAGE-E-BADVALUE, illegal qualifier value

Explanation: You have specified a qualifier value that is invalid.

User Action: Reenter the statement with a valid qualifier value.

MESSAGE-E-CONFFAC, facility definition conflicts with previous definition

Explanation: You have specified the same facility name or facility number for two different facilities.

User Action: Reenter the facility definitions so that each facility name corresponds to one facility number.

MESSAGE-E-DATAOVFL, data area overflow in section - please submit SPR

Explanation: An internal error in the compiler has been detected.

User Action: Submit a Software Performance Report (SPR) to DIGITAL, describing a situation.

MESSAGE-E-DUPMSG, message code nn already assigned as ss

Explanation: You have assigned the same message code, (here represented by nn) to more than one message symbol. The first message symbol assigned to the message code is shown (here represented by ss).

User Action: Check that you have not assigned the same message number or symbol name twice for the same facility. Reenter the message definition(s) so that each message symbol is assigned to one message code.

MESSAGE-E-DUPSYM, duplicate symbol definition

Explanation: You have assigned the same symbol to represent two values. This message may refer to symbols defined in a message definition, a facility definition, or the .LITERAL directive.

User Action: Reenter your source statements with each symbol defined only once.

MESSAGE-E-FACOVFL, facility table overflow in section - please submit SPR

Explanation: An internal error in the compiler has been detected.

User Action: Submit a Software Performance Report (SPR) to DIGITAL, describing the situation.

MESSAGE-E-INDEXOVFL, index area overflow in section - please submit SPR

Explanation: An internal error in the compiler has been detected.

User Action: Submit a Software Performance Report (SPR) to DIGITAL, describing the situation.

MESSAGE-E-NOMSGS, no messages defined

Explanation: No messages have been defined in the message source file.

User Action: Ensure that at least one message definition appears in the message source file.

MESSAGE-E-NOSEVER, severity unspecified, ERROR used

Explanation: You have omitted defining the severity of a facility's messages. The Message Utility has assigned the level ERROR to the messages.

User Action: Add the appropriate severity levels to the message source file, either with severity definitions or with severity qualifiers on message definitions.

MESSAGE-E-SHARCONF, /SHARED conflicts with facility number

Explanation: You have specified the /SHARED qualifier on the facility definition with a facility number other than 0.

User Action: Remove the /SHARED qualifier from the facility definition.

MESSAGE-E-SYMTOOLNG, symbol name too long

Explanation: You have assigned a symbol name with more characters than are permitted.

User Action: Shorten the symbol name.

APPENDIX A

FILES-11 DEVICES SUPPORTED BY VAX/VMS

Tables A-1 and A-2 list the Files-11 structured devices supported by VAX/VMS, with their storage characteristics and the device codes used to refer to them. Table A-1 lists magnetic tape devices; Table A-2 lists disk devices. See the VAX/VMS I/O User's Guide for more information on these devices.

Table A-1
Magnetic Tape Devices

Model	Code	No. of Tracks	Recording density (bpi)	Tape speed (ips)	Max. data transfer rate in bytes per second	Recording method
TE16	MT	9	800 or 1600	45	36,000 (for 800 bpi); 72,000 (for 1600 bpi)	NRZI or PE[1]
TS11	MS	9	800 or 1600	45	36,000 (for 800 bpi); 72,000 (for 1600 bpi)	NRZI or PE
TU45	MT	9	800 or 1600	75	60,000 (for 800 bpi) 120,000 (for 1600 bpi)	NRZI or PE
TU77	MT	9	800 or 1600	125	100,000 (for 800 bpi) 200,000 (for 1600 bpi)	NRZI or PE

1. NRZI = non-return-to-zero-inverted; PE = phase encoded.

Table A-2
Disk Devices

Model	Code	Type[1]	RPM	Surfaces	Cylinders	Bytes/ Track	Bytes/ Drive
RL02	DL	Cart	2400	2	512	10,240	10,485,760
RM03	DR	Pack	3600	5	823	16,384	67,420,160
RP05	DB	Pack	3600	19	411	11,264	87,960,576
RP06	DB	Pack	3600	19	815	11,264	174,423,040
RK07	DM	Cart	2400	3	815	11,264	27,550,480
RX01	DX	Flop	360	1	77	3,328	256,256
RX02	DY	Flop	360	1	77	3,328[2] 6,656[3]	256,256[2] 512,512[3]
TU58[4]	DD	Cart	##	##	##	##	262,144

1. Pack = pack disk; Cart = cartridge disk; Flop = floppy (flexible diskette)

2. Single density

3. Double density

4. A magnetic tape device, the TU58 operationally resembles a disk device

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

Please cut along this line.

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in t
United Stat

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here