RSTS/E
Programmer's Utilities Manual

Order No. AA–D749A–TC

Including AD–D749A–T1, T2

June 1982

This document describes the RSX–based utilities available to the RSTS/E programmer. It contains information on the MACRO Assembler, Librarian, Patch, and MAKSIL utilities.

OPERATING SYSTEM AND VERSION:	RSTS/E	V7.2
SOFTWARE VERSION:	RSTS/E	V7.2

digital equipment corporation, maynard, massachusetts

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

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

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

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

The following are trademarks of Digital Equipment Corporation:

DEC	VT	IAS
DECUS	DECsystem–10	MASSBUS
DECnet	DECSYSTEM 20	PDT
PDP	DECwriter	RSTS
UNIBUS	DIBOL	RSX
VAX	Edusystem	VMS
		digital

Commercial Engineering Publications typeset this manual using DIGITAL's TMS–11 Text Management System.

Contents

Page

Preface *vii*

Documentation Conventions *viii*

Chapter 1 Introduction

1.1 RSTS/E Utility Command Line 1–1
1.2 RSTS/E File Specifications 1–2
1.3 Accessing Utilities and Entering Command Lines 1–3
1.3.1 Accessing Utilities 1–3
1.3.1.1 Entering the RUN Command 1–4
1.3.1.2 CCL Command Names 1–4
1.3.2 Entering Command Lines 1–5
1.3.2.1 Entering the Complete Command Lines 1–5
1.3.2.2 Using Indirect Command Files 1–5
1.3.2.3 Using Continuation Lines 1–6

Chapter 2 Using the MACRO–11 Utility Program

2.1 Running the MACRO–11 Assembler 2–2
2.1.1 Running MACRO in DCL 2–2
2.1.2 Running MACRO with the RUN Command or a CCL Command 2–4
2.1.3 Using Indirect Command Files 2–5
2.2 File Specification Switches 2–6
2.2.1 Listing Control Switches 2–7
2.2.2 Function Control Switches 2–8
2.2.3 MACRO Library Switch 2–9
2.2.4 Assembly Pass Switch 2–9
2.3 Error Messages 2–10

Chapter 3 Using the Librarian Utility Program (LBR)

3.1 Library Files 3–1
3.2 LBR Command Line 3–2
3.3 LBR Switches 3–2
3.3.1 Compress Switch (/CO) 3–3
3.3.2 Create Switch (/CR) 3–4
3.3.3 Delete Switch (/DE) 3–5
3.3.4 Default Switch (/DF) 3–6
3.3.5 Delete Global Switch (/DG) 3–8
3.3.6 Entry Point Switch (/EP) 3–8
3.3.7 Extract Switch (/EX) 3–10
3.3.8 Insert Switch (/IN) for Object and Macro Libraries 3–11
3.3.9 Insert Switch (/IN) for Universal Libraries 3–12

3.3.10 List Switches (/LI, /LE, /FU) . . . 3–13
3.3.11 Modify Header Switch (/MH) . . . 3–14
3.3.12 Replace Switch (/RP) for Object and Macro Libraries . . . 3–15
3.3.13 Replace Switch (/RP) for Universal Libraries . . . 3–18
3.3.14 Selective Search Switch (/SS) . . . 3–20
3.3.15 Spool Switch (/SP) . . . 3–20
3.3.16 Squeeze Switch (/SZ) . . . 3–21
3.4 Combining Library Functions . . . 3–23
3.5 LBR Restrictions . . . 3–23
3.6 LBR Error Messages . . . 3–24
3.6.1 Effect of Fatal Errors on Library Files . . . 3–24
3.6.2 List of LBR Errors . . . 3–25

Chapter 4 Using the Object Module Patch Utility (PAT) Program

4.1 How PAT Works . . . 4–2
4.2 Specifying the PAT Command Line . . . 4–4
4.3 How PAT Applies Updates . . . 4–4
4.3.1 The Input File . . . 4–4
4.3.2 The Correction File . . . 4–5
4.3.3 Creating the Correction File . . . 4–5
4.3.4 How PAT and the Task Builder Update Object Modules . . . 4–6
4.3.4.1 Overlaying Lines in a Module . . . 4–6
4.3.4.2 Adding a Subroutine to a Module . . . 4–7
4.3.5 Determining and Validating the Contents of a File . . . 4–8
4.4 PAT Messages . . . 4–9
4.4.1 Information Messages . . . 4–9
4.4.2 Command Line Errors . . . 4–10
4.4.3 File Specification Errors . . . 4–11
4.4.4 Input/Output Errors . . . 4–12
4.4.5 Errors in File Contents or Format . . . 4–13
4.4.6 Internal Software Error . . . 4–14
4.4.7 Storage Allocation Error . . . 4–15

Chapter 5 Using the MAKSIL Utility Program

5.1 Creating a Run-Time System (RTS) . . . 5–1
5.2 Creating a Resident Library . . . 5–3
5.3 Operating Instructions . . . 5–3
5.4 Messages . . . 5–5
5.4.1 Fatal Error Messages . . . 5–5
5.4.2 Diagnostic Messages . . . 5–8
5.4.3 Informational Messages . . . 5–9

Appendix A MACRO–11 Diagnostic Error Message Summary

Appendix B Librarian Utility Program (LBR) Files and Formats

B.1 Library Header B–1
B.2 Entry Point Table B–1
B.3 Module Name Table B–1
B.4 Module Header B–2

Index

Figures

3–1 MACRO Listing Before and After Running LBR with /SZ Switch . . 3–22
4–1 Updating a Module Using PAT 4–2
4–2 Processing Steps Required to Update a Module Using PAT. 4–3
B–1 Standard Library File Format B–2
B–2 Universal Library File Format B–3
B–3 Contents of Library Header B–4
B–4 Format of Entry Point Table Element B–5
B–5 Format of Module Name Table Element B–5
B–6 Module Header Format B–5

Tables

1–1 File Specification Defaults. 1–2
1–2 File Extension Defaults 1–3
1–3 Conventional CCL Names for RSTS/E Accessed RSX–Based Utilities . 1–4
2–1 MACRO–11 Switches 2–6
2–2 Arguments for /LI and /NL Switches 2–7
2–3 Arguments for /DS and /EN Switches. 2–8
3–1 LBR Switches 3–2
3–2 Sample Files Used in LBR Examples 3–16
3–3 Output Library File after Execution of Example 1 3–17
3–4 Output Library File after Execution of Example 2 3–17
3–5 Output Library File after Execution of Example 3 3–18
5–1 Task Builder Options for Virtual and Physical Address Range 5–2
5–2 Task Builder PAR and STACK Options for Various Sized Run-Time Systems. 5–2

Preface

This manual describes four RSX–based utility programs available on RSTS/E: MAC, LBR, PAT, and MAKSIL.

- MAC – The MACRO–11 Assembler processes assembly language programs and produces single relocatable binary object files.
- LBR – The Librarian Utility is a library maintenance program that provides a means for creating, modifying, updating, listing, extracting, and maintaining library files.
- PAT – The Object Module Patch Utility is used to modify code in a relocatable binary object module.
- MAKSIL – The Make a Save Image Library Utility is used to create a resident library, a run-time system, or a multi-user task.

Audience Description

To use this manual, you should be familiar with the MACRO computer language and have a working knowledge of the RSTS/E operating system.

Document Structure

This manual has five chapters and two appendixes:

- Chapter 1 Introduction

  Introduces you to the RSX–11M utilities as they are used on a RSTS/E system.

- Chapter 2 Using the MACRO–11 Utility Program

  Describes how to run the MACRO assembler to convert MACRO source code into object code.

- Chapter 3 Using the Librarian Utility Program (LBR)

  Shows how to create, modify, maintain, and use library files containing MACRO modules.

- Chapter 4 Using the Object Module Patch Utility Program (PAT)

  Shows how to modify code in a relocatable binary object module.

- Chapter 5 Using the MAKSIL Utility Program

  Describes how to create a resident library, a run-time system, or a multi-user task.

- Appendix A MACRO–11 Diagnostic Error Message Summary

  Describes the single character codes that identify MACRO programming errors. (Appendix A does not, however, contain a description of the MACRO input-output error messages. You must refer to Chapter 2 for that information.)

- Appendix B Librarian Utility Program (LBR) Files and Formats

  Contains detailed information on the formats and contents of library files.

Associated Documents

The *PDP–11 MACRO–11 Language Reference Manual* describes how to use the MACRO–11 relocatable assembler to develop PDP–11 assembly language programs.

The *RSTS/E DCL User's Guide* describes the use of DCL (DIGITAL Command Language) on RSTS/E.

See the *RSTS/E Documentation Directory* for more information on RSTS/E manuals.

Documentation Conventions

This manual uses the following conventions:

RET	This symbol represents a carriage return. Unless the manual indicates otherwise, end all commands or command strings with a carriage return.
@	This symbol invokes an indirect command file. The at sign immediately precedes the file specification for an indirect command file.
(/)	Slashes in the file specification precede switches or qualifiers.
(.)	Periods in the file specification separate the file name and extension. When the file name is used without an extension, the period is not necessary.
UPPERCASE	In discussions of syntax, uppercase letters represent the command name, which you must type.
lowercase	Lowercase letters represent a variable, for which you must supply a value.
[ ]	Square brackets enclose an item that is optional. You may include the item in brackets, or you may omit it, as you choose.
(...)	The ellipsis symbol represents repetition. You can repeat the item that precedes the ellipsis.

Chapter 1
Introduction

This chapter describes the following subjects:

1. The RSTS/E command line.
2. The RSTS/E file specification.
3. The use of utilities and the entering of command lines.
4. The conditions under which you can use a utility.

These subjects are common to all the programmer's utilities described in this manual.

1.1 RSTS/E Utility Command Line

The general utility command line format is:

```
outfile,...outfile=infile,...infile
```

where outfile and infile are file specifications for the output and input files to be operated on by the utility. The number of file specifications you can enter depends on the utility invoked. The maximum length of a command line is 80 characters for all utilities except MACRO, which sometimes allows up to 132 characters. (See Section 2.2.)

This general format varies for each utility. Some utilities use the entire command line and others use abbreviated forms of the command line. These utilities also accept indirect files containing command lines, as described in Section 1.3.2.2.

1.2 RSTS/E File Specifications

A file specification consists of a filename that conforms to standard RSTS/E conventions, plus switches that modify, or specialize, the command. File specifications have the form:

```
device:[project,programmer]filename.extension/sw...
```

where all components are optional except the filename. The file specification components are defined below:

device	is the name of the device that stores the file. The device name consists of two ASCII characters followed by an optional 1–or 2–digit decimal unit number; for example, LP or DT1. Logical device names of up to six alphanumeric characters may also be used.
[project, programmer]	is the account or project-programmer number (PPN) associated with the file. The default is the PPN of the account you have logged into. Note that RSTS/E project-programmer numbers are similar to RSX–user identification codes (UIC).
filename	is the name of the desired file. The filename can contain up to six alphanumeric characters.
extension	is the 0– to 3–character filename extension. Files having the same name but a different function can be distinguished from one another by the file extension; for example, LRB.TSK and LRB.OBJ.
/sw	is a switch specification. More than one switch can be used, each separated from the previous one by its slash (/). The switch name is a 2– to 4–character alphanumeric code that identifies the switch and may also indicate negation of the switch. The permissible switches and their syntax are presented for each utility in the pertinent chapter.

You can use RSTS/E–specific switches (such as /MODE and /SIZE) only with the MAKSIL utility.

Table 1–1 lists the default assumptions for components of a file-specification that are not designated.

Table 1–1: File Specification Defaults

Item	Default
device	The device last specified (SY:, if none).
project-programmer	The project-programmer number last specified (the account you have logged into, if there is no previous entry). Any PPN associated with a previously specified logical device name does not carry through.
extension	See Table 1–2.
switch	Defaults for each utility described in Chapters 2 through 5.

Following is an example of input to the MAC Assembler and defaults:

```
DK1:IMG1,MP1=IN1,DB0:IN2,IN3
```

Device	File
DK1:	IMG1.OBJ
DK1:	MP1.LST
SY:	IN1.MAC
DB0:	IN2.MAC
DB0:	IN3.MAC

Table 1–2 lists the default assumptions for missing extensions.

Table 1–2: File Extension Defaults

Utility	File Type	Extension	File Description
MACRO–11 (MAC)	Input	.CMD	Indirect Command File
		.MAC	Macro Module
		.MLB	Macro Module Library
	Output	.OBJ	Object Module
		.LST	List File
Library (LBR)	Input	.CMD	Indirect Command File
	Input or Output	.OBJ	Object Module
	Input or Output	.MAC	Macro Source Module
	Input or Output	.OLB	Object Library Module
	Input or Output	.MLB	Macro Module Library
	Output	.LST	List File
Patch (PAT)	Input	.CMD	Indirect Command File
		.OBJ	Object Module
	Output	.OBJ	Object Module
MAKSIL	Input	.TSK	Task Image
		.STB	Symbol Table File
		.CMD	Indirect Command File
	Output	.LIB	Resident Library File
		.RTS	Run-Time System File
		.CMD	Indirect Command File

1.3 Accessing Utilities and Entering Command Lines

The RSTS/E user can access an RSX utility in two ways, and after invoking a specific utility, the user then has two choices for entering command lines. The paragraphs that follow describe the methods of accessing utilities and entering command lines, respectively.

1.3.1 Accessing Utilities

The two ways to invoke a utility are:

1. Type the RUN command.
2. Type a CCL command.

The paragraphs below describe each method.

1.3.1.1 Entering the RUN Command — In response to the system READY prompt, you can enter the general form of the RUN command:

```
RUN $utility RET
```

where utility is one of the following:

MAC.TSK	– MACRO–11 Assembler Utility
LBR.TSK	– Librarian Utility
PAT.TSK	– Patch Object Module Utility
MAKSIL.BAC	– MAKSIL Utility

For example, you can invoke the Librarian utility by typing the following:

```
RUN $LBR RET
```

This prompt is displayed on the terminal to indicate the Librarian utility is ready to accept a command line:

```
LBR>
```

Note that the use of the symbol $ indicates that the utility is stored in account [1,2]. The system manager has the option of installing these utilities in other accounts. Check with the system manager for the location of these utilities.

1.3.1.2 CCL Command Names — If the system manager has installed CCL commands for the programmer's utilities, you can invoke these utilities by using the appropriate CCL command. Table 1–3 lists a recommended set of CCL names for the RSX utilities invoked by RSTS/E.

Table 1–3: Conventional CCL Names for RSX–Based Utilities

Utility	CCL Name
MACRO–11	MAC
Librarian	LBR
Patch Object Module	PAT
Make Save Image Library	MAKSIL

As an example, you can invoke the PAT utility by typing:

```
PAT RET
```

The utility indicates its readiness to accept a command line by displaying the following prompt:

```
PAT>
```

Alternatively, you can employ the following general form of the CCL command:

```
PAT <command-line> RET
```

This form causes the Patch utility to run, process <command-line>, and return to the the system prompt.

Some utilities can also be invoked by DCL commands. See the *RSTS/E DCL User's Guide* for more information.

1.3.2 Entering Command Lines

The two methods for entering utility command lines are:

1. Typing the complete utility command line.
2. Using an indirect command.

1.3.2.1 Entering the Complete Command Lines — You can enter the required command line either in response to a utility's prompt for input or as part of the CCL command. Three examples of this method are:

1. You can employ the RUN command to invoke the utility. The utility prompts for command line input. After execution is completed, the utility reprompts for additional command input; for example:

   ```
   RUN $LBR RET
   LBR>BIGLIB/IN=SMALL,MID,BIG RET
   LBR>
   ```

2. You can invoke the utility by typing its CCL name. The utility prompts for the command line. After execution is completed, the utility reprompts for additional command input; for example:

   ```
   LBR RET
   LBR>BIGLIB/IN=SMALL,MID,BIG RET
   LBR>
   ```

3. You can invoke the utility by typing its CCL name followed by a space and the complete utility command line. After execution of the utility, the system again displays the READY prompt; for example:

   ```
   LBR BIGLIB/IN=SMALL,MID,BIG RET
   READY
   ```

1.3.2.2 Using Indirect Command Files — The second method of entering a utility command line is through the use of indirect command files. When you specify an indirect command file, the utility interprets the contents of the file in the command specified as a series of one or more command lines. The advantage of an indirect command file is that you can enter a commonly used command line sequence once and store it for subsequent use rather than reentering the sequence.

The @ character is the first character of the indirect command line. Immediately following the @ character is a file specification. The format for an indirect command is:

```
@device:[project,programmer] filename.extension
```

You can omit certain elements of the file specification. The following defaults are then applied:

device	– SY:
[project,programmer]	– Current PPN
.extension	– .CMD

The following examples show the use of indirect commands.

Example 1

```
RUN $LBR (RET)
LBR>@ALPHA (RET)
LBR>
```

Example 2

```
LBR @BETA.CTL (RET)
READY
```

In the first example, only the filename, ALPHA, is specified. The device, account, and extension fields are defaulted. In the second example, a filename and extension are specified with the device and account defaulted.

NOTE

Indirect command files are not used by the MAKSIL program.

1.3.2.3 Using Continuation Lines — Only the MACRO utility allows you to use continuation lines. See Chapter 2 for more information.

Chapter 2
Using the MACRO–11 Utility Program

RSTS/E has two MACRO language assemblers, one that runs under the RT11 run-time system and one that runs under the RSX–11M run-time system. This chapter describes how to use the RSX–based MACRO assembler. Refer to the *PDP–11 MACRO–11 Language Reference Manual* for complete information about MACRO–11 assembly language and the *RSTS/E RT11 Utilities Manual* for information on the RT11–based MACRO assembler.

The RSX–based MACRO assembler (MAC) converts MACRO source code into object code. Output from the assembler can include any or all of the following:

- A binary relocatable object file that contains all the records and relocation information needed for task building
- A listing of the source input file that provides both documentation for the module and a tool for debugging the code
- A table of contents listing that contains the line sequence numbers, the page numbers, and the text accompanying each .SBTTL directive
- A symbol table listing that provides information about symbol names and program sections (.PSECTs) that are referenced in the source program

To use the MACRO assembler, you should understand how to:

- Run the assembler (including how to format command strings to specify files MACRO uses during assembly)
- Use file specification switches to override file control directives in the source program
- Interpret error messages

The following sections describe these topics.

2.1 Running the MACRO Assembler

There are two ways to run the MACRO assembler on RSTS/E:

- With the DIGITAL Command Language (DCL) MACRO command
- With the RUN command or a Concise Command Language (CCL) command

You can also use indirect command files to enter command lines.

Use the method that best suits your needs and the conditions on your system (default keyboard monitor, for example).

2.1.1 Running MACRO in DCL

You can recognize DCL by its dollar prompt ($). To run MACRO, type:

```
$ MACRO RET
```

After you press RETURN, DCL prompts for the input files:

```
Files:
```

Enter the input files in the form:

input filespec + ... + n

For example:

```
Files:     FILE1+FILE2+FILE3 RET
```

In this example FILE1, FILE2, and FILE3 are the source files to be assembled.

You can also type a complete command on one line. For example:

```
$ MACRO MAIN.MAC RET
```

MACRO creates the object file MAIN.OBJ and the list file MAIN.LST from the source file MAIN.MAC and places the files in your account on the public structure.

Note that you cannot use the wildcard characters (?) and (*) with the DCL MACRO command.

The DCL processor expands the file specifications you enter into more complete file specifications, including device and account. The expanded command line can contain no more than 80 characters. If DCL detects more than 80 characters in the expanded command line, you receive the error message "?Command too long." Note that you can use a plus sign (+) or a comma (,) to separate the input files.

By default, MACRO creates an object file and gives it the same name as the first input file. The assembler also creates a list file by default. The default file extensions are .OBJ for the object file and .LST for the list file.

Although an input file can be on a disk other than the system disk, MACRO places the object and list files in your account on the public structure by default. You can include a device specification in the command line directing MACRO to place the files on a specified device.

If you do not want MACRO to create an object file, use the /NOOBJ qualifier. Similarly, the /NOLIST qualifier tells MACRO not to create a list file. You use these qualifiers when you want to see if a MACRO program assembles, and you do not want to get an object or a list file (at least the first time you run the program).

To give the object or list file a name other than the default, use the /OBJECT or /LIST qualifier to assign a specific file name. If you do not include file extensions, the assembler assigns them for you. For example:

```
$ MACRO MAIN.MAC/OBJECT=MODULE/LIST=MODLST RET
```

In this example:

MACRO	Invokes the assembler.
MAIN.MAC	Is the source input file.
/OBJECT=MODULE	Specifies MODULE as the object file name.
/LIST=MODLST	Specifies MODLST as the list file name.

You can also specify a library file. To do so, add the /LIBRARY qualifier to the input file specification. For example:

```
$ MACRO MAIN.MAC+SECOND.LIB/LIBRARY/OBJECT=MODULE RET
```

In this example:

MACRO	Invokes the assembler.
MAIN.MAC	Is a source input file.
SECOND.LIB	Is a second source input file.
/LIBRARY	Is the library qualifier that marks the input file SECOND.LIB as a library file created by the LBR program.
/OBJECT=MODULE	Specifies MODULE as the object file name.

If you are at the Files: prompt and want to exit the MACRO assembler, press CTRL/Z to return to the dollar prompt. If you have begun an assembly, you can stop the process and return to the dollar prompt by pressing CTRL/C. Note that CTRL/C is an abnormal exit that you should use only as a last resort.

Refer to the *RSTS/E DCL User's Guide* for complete information about DCL.

2.1.2 Running MACRO with the RUN Command or a CCL Command

On most RSTS/E systems, the MACRO assembler runs from the system library account [1,2]. You can use the RUN command to run MACRO from any keyboard monitor by typing:

```
RUN $MAC (RET)
```

The $ is a special account character representing [1,2].

Your system manager may have installed a Concise Command Language (CCL) command to run MACRO. If so, you can type:

```
MAC (RET)
```

In response, MACRO prompts with:

```
MAC>
```

Note that in DCL, typing MAC causes the Files: prompt to appear on your terminal. To use the CCL MAC in DCL, type:

```
$ CCL MAC (RET)
```

The MAC> prompt indicates that the assembler is ready for a command of the form:

output-filespec(s) = input filespec(s)

The file specifications define the object files, listing files and source files appearing in a MACRO command line.

Type output and input file specifications in the form:

object,listing/s:arg = src1,src2....srcn/s:arg

where:

object	Is the file specification of the binary object file produced by the assembly process.
listing	Is the file specification of the assembly and symbol listing produced by the assembly process.
/s:arg	Is one or more file specification switches and arguments. (Section 2.3 describes these switches and arguments.)
src1,src2....srcn	Are file specifications for one or more MACRO source files or MACRO library files.

You cannot use wildcard characters (?) and (*) in MACRO file specifications.

If you omit the object or listing output file specification, the output file is not created. For example:

```
MAC>,LIST=SRC1,SRC2 RET
```

In this case an object file is not created, but the list file is. Note that you must include the comma before "LIST".

If you do not enter an input file, you receive the error message "MAC – ILLEGAL FILENAME."

If you do not specify a listing file, MACRO displays on your terminal any errors found in the source program.

You can enter a complete command on one line when you use the CCL command, but you are limited to 80 characters. However, when you are at the MAC> prompt, you can have a total of 132 characters in a command line.

You can use continuation lines at the MAC> prompt, as long as the total length does not exceed 132 characters. A continuation line allows you to type a command line on more than one physical line. Use a hyphen to indicate continuation after any element in the command line. For example:

```
MAC>OBJECT,LISTING=SRC1,- RET
MAC>SRC2,SRC3,- RET
MAC>SRC4 RET
```

If you are at the MAC> prompt and want to exit the MACRO assembler, press CTRL/Z to return to the keyboard monitor prompt. If you have entered a command string and begun an assembly, you can stop the process and return to the keyboard monitor prompt by pressing CTRL/C.

2.1.3 Using Indirect Command Files

Besides typing command lines using the methods described in Sections 2.1.1 and 2.1.2, you can also enter command lines by using an indirect command file. When you specify an indirect command file, MACRO interprets the contents of the file as one or more command lines. The advantage of an indirect command file is that you can enter a commonly used command sequence once and store it for later use.

You can use EDT or any other editor to create the indirect command file. For example, the contents of an indirect command file might be:

```
MAIN,MAIN=MAIN
MACRO,MACRO=MACRO
```

Here is an example that shows the use of an indirect command file:

```
RUN $MAC RET
MAC>@filespec RET
MAC>
```

The file specified as "@filespec" contains MACRO command strings. After opening this file, MACRO reads and executes command lines until it detects the end of file. MACRO then displays the MAC> prompt. Press CTRL/Z to exit.

You can also use indirect command files in a CCL command:

```
MAC @filespec RET
```

MACRO reads and executes command lines until it detects the end of file. Control then returns to the keyboard monitor.

Note that you cannot use indirect command files with the DCL MACRO command.

You can nest indirect command files in MACRO (that is, call one indirect command file from another). However, you cannot nest command files to more than three levels. If you do, you receive the error message "MAC — INDIRECT FILE DEPTH EXCEEDED."

2.2 File Specification Switches

At assembly time, you can override specific MACRO directives that appear in the source program. You can also tell MACRO how to handle specific files during assembly. You do this by including special switches in the MACRO command string. Table 2–1 lists the switches and describes their effects.

Table 2–1: MACRO–11 Switches

Switch	Type	Function
/LI:arg	Listing Control	Overrides source program directive .NLIST.
/NL:arg	Listing Control	Overrides source program directive .LIST.
/SP	Listing Control	Spools listing output.
/NOSP	Listing Control	Does not spool output (default value).
/EN:arg	Function Control	Allows the use of arguments (see Table 2–3) to override source program directive .DSABL.
/DS:arg	Function Control	Allows the use of arguments (see Table 2–3) to override source program directive .ENABL.
/ML	MACRO Library	Indicates that the input file is a MACRO library file.
/PA:1	Assembly Pass	MACRO is a 2–pass assembler. /PA:1 causes the associated file to assemble during pass 1 only.
/PA:2	Assembly Pass	Causes the associated file to assemble during pass 2 only.

Attach the /ML, /PA:1 and /PA:2 switches directly to the source files they affect. Place other switches anywhere in the command string. For example, the /LI switch affects the listing file regardless of where you place it in the command string.

The next four sections describe how to use the file specification switches.

2.2.1 Listing Control Switches

Use the /LI:arg and /NL:arg switches to:

- Control the content and format of assembly listings at assembly time
- Override the arguments of .LIST and .NLIST directives in the source program

MACRO has default settings for the /LI and /NL switches when you do not include arguments:

- The /LI switch without an argument causes MACRO to ignore .LIST and .NLIST directives that have no arguments
- The /NL switch without an argument causes MACRO to list only the symbol table, the table of contents, and error messages

Table 2–2 lists the arguments you can use with /LI and /NL switches. See the *PDP–11 MACRO–11 Language Reference Manual* for more information on the meaning of these arguments.

Table 2–2: Arguments for /LI and /NL Switches

Argument	Listing Control	Default
BEX	Binary extensions.	List
BIN	Generated binary code.	List
CND	Unsatisfied conditionals, .IF and .ENDC statements.	List
COM	Comments.	List
LD	List control directives with no arguments.	No List
LOC	Address location counter.	List
MC	Macro calls, repeat range expansion.	List
MD	Macro definitions, repeat range expansion.	List
ME	Macro expansion.	No List
MEB	Macro expansion binary code.	No List
SEQ	Source line sequence number.	List
SRC	Source code.	List
SYM	Symbol table.	List
TOC	Table of contents.	List
TTM	132–column line printer format when not specified.	No List

For example, assume that the source program contains the following sequence:

```
.NLIST MEB
       .
       . (MACRO references)
       .
.LIST  MEB
```

In this example, you disable the listing of MEB (Macro Expansion Binary) code with the .NLIST directive and later resume MEB listing with the .LIST directive. If you include /LI:MEB in the assembly command string, however, MACRO ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

2.2.2 Function Control Switches

The /EN:arg and /DS:arg switches let you enable or disable functions at assembly time and control the form and content of the binary object file. These switches override .ENABL and .DISABL directives in the source program.

Table 2–3 summarizes the /EN and /DS function arguments, their default status, and the functions they control. See the *PDP–11 MACRO–11 Language Reference Manual* for more information on the meaning of the arguments.

Table 2–3: Arguments for /EN and /DS switches

Argument	Function	Default
ABS	Absolute binary output.	Disable
AMA	Assembly of all absolute addresses as relative addresses.	Disable
CDR	Source columns from 73 on are reserved for comments.	Disable
FPT	Floating point truncation.	Disable
GBL	Undefined symbols treated as globals.	Disable
LC	Lowercase ASCII input.	Disable
LSB	Local symbol block.	Disable
PNC	Binary output.	Enable
REG	Mnemonic definitions of registers. MACRO uses the following definitions: R0 = %0 R1 = %1 R2 = %2 R3 = %3 R4 = %4 R5 = %5 SP = %6 PC = %7	Enable

2.2.3 Macro Library Switch

Before using the macro library switch, you need to understand what macro libraries are and how the assembler and Task Builder handle them. Macro libraries are specially formatted files containing macro call definitions or object module subroutines. The MACRO assembler and the Task Builder can access these library files and extract code from them. Libraries are convenient to use because they encourage sharing of code.

RSTS/E comes with a set of standard libraries. You can also create your own libraries with LBR, described in Chapter 3. The library files you create using LBR are in the same format as those that DIGITAL supplies with the operating system.

LB:RSXMAC.SML is one of the libraries that DIGITAL provides. It contains definitions for all system directives. You do not have to specify LB:RSXMAC.SML in the command string; MACRO automatically searches it as if it were the last source file in the command string. However, if you include a macro library in the command string other than LB:RSXMAC.SML, macro definitions contained in that library override the macro definitions in LB:RSXMAC.SML.

Use the /ML switch to designate a source file as a macro library. The /ML switch specifies that macros referenced in the .MCALL directive come from a library other than LB:RSXMAC.SML. When MACRO encounters the /ML switch, it searches the designated library first. If the macro definition is not found in the designated library, MACRO then searches LB:RSXMAC.SML.

When MACRO locates an .MCALL directive in the source code, it searches macro libraries that are named in the command string in reverse order. Thus, if two or more macro libraries contain definitions of the same macro name, the macro library that appears last in the command string takes precedence. For example:

MAC><output file specification>=ALIB.MLB/ML,BLIB.MLB/ML,XIZ

Assume that each of the two macro libraries, ALIB and BLIB, contains a macro called .BIG, but with different definitions. If source file XIZ contains a macro call .MCALL .BIG, the system includes in the program the definition of .BIG that appears in the macro library BLIB.

2.2.4 Assembly Pass Switch

Use the /PA:arg switch to specify whether the source file assembles in pass 1 or pass 2. The /PA:arg switch is meaningful only if you add it to a source input file specification.

The specification /PA:1 calls for assembly of the file during pass 1 only. During the first pass, the assembler groups all symbols as either local or global, performs statement generation, locates all macro symbols, and, if necessary, reads the macro definitions from libraries. At the end of pass 1, the assembler will have processed all local references (such as undefined global symbols) that are to be resolved by the Task Builder.

Use the /PA:1 switch for files that assemble completely at the end of pass 1. Definition files (prefix files containing only symbol definitions) are a good example. By specifying /PA:1 for these files, you can cause MACRO to skip processing of these files in pass 2. In some cases, this procedure can save considerable assembly time.

If you have conditional assemblies or use .PSECTs in your program, do not use the /PA:1 switch. Using /PA:1 in these cases can cause errors in the table of contents listing or can result in link time errors. The next two paragraphs provide more details about these errors.

The table of contents listing may contain inaccurate page numbers if a portion of a program is not assembled during pass 2 (using the /PA:1 switch). This is because page numbers are sent to the listing file during pass 1, while listing pages are sent to the listing file during pass 2. Page numbering may change as the result of conditional assemblies (assembled during pass 2), but these new page numbers will not appear in the table of contents listing.

You must also be careful in declaring .PSECTs, because the GSD (Global Symbol Definition) portion of the .OBJ file is output at the end of pass 1. If a portion of your program redefines a .PSECT, or changes the size of a .PSECT, this will not be reflected in the .OBJ file and will result in link time errors.

The specification /PA:2 calls for assembly of the file during pass 2 only. During the second pass, the assembler actually generates the object module and listing files, flagging with an error code in the listing file those source statements containing errors. This switch (/PA:2), though not very useful, is provided as part of the syntax.

2.3 Error Messages

MACRO can detect two types of errors: input-output and programming. Input-output errors occur when you specify incorrect command strings to MACRO or when problems arise with I/O devices. These errors appear on your terminal when you assemble your program. Programming errors are mistakes in source code syntax or faulty program logic. These codes automatically appear on the assembly listings.

This section describes input-output errors; see Appendix A for a description of the single character codes that identify MACRO programming errors.

All the error messages listed here, except for the "MAC — COMMAND I/O ERROR" message, cause the current assembly to end. MACRO then tries to restart by reading another command line. If a command I/O error occurs, MACRO exits because it cannot get additional command line input.

```
MAC -- COMMAND FILE OPEN FAILURE
```

Description

MACRO cannot open the indirect command file. See "OPEN FAILURE ON INPUT FILE" for meaning.

Suggested Response

Make sure the command file exists and you have read access to the file. Check spelling errors in your command line.

```
MAC -- COMMAND I/O ERROR
```

Description

The file system detects an error during MACRO's attempt to read a command line. This is a fatal error and causes MACRO to exit. The device may be off line or write-protected, or a bad block may exist in the file.

Suggested Response

Check the command file for correct contents. If the indirect command file contains a bad disk block, re-create the file.

```
MAC -- COMMAND SYNTAX ERROR
```

Description

MACRO detects an error in the syntax of the command line.

Suggested Response

Check the command line for spelling errors. Make sure you have specified an input file. Check all switches for correct spelling and correct arguments.

```
MAC -- ILLEGAL FILENAME
```

Description

The input file specification is missing or the input or output file specification contains an illegal character. You cannot use the wildcard characters (?) and (*) in MACRO file specifications.

Suggested Response

Check for spelling errors in the command line. Check file specifications for correct format.

```
MAC -- ILLEGAL SWITCH
```

Description

MACRO does not recognize the specified switch.

Suggested Response

Check for spelling errors in the command line. Make sure all switch arguments are legal.

```
MAC -- INDIRECT COMMAND SYNTAX ERROR
```

Description

The name of the indirect command file specified in the MACRO command line is syntactically incorrect.

Suggested Response

Check for spelling errors in the command line. Make sure there are no wildcards in your file specification.

```
MAC -- INDIRECT FILE DEPTH EXCEEDED
```

Description

Indirect command files are nested to more than three levels.

Suggested Response

Restructure the command files so they are not nested to more than three levels.

```
MAC -- INSUFFICIENT DYNAMIC MEMORY
```

Description

The assembler has run out of space.

Suggested Response

Segment your program and assemble the pieces or have the system manager increase the system's swap maximum.

```
MAC -- INVALID FORMAT IN MACRO LIBRARY
```

Description

The library file is corrupt (contains bad data) or the Librarian Utility Program (LBR) did not create it.

Suggested Response

Make sure you are using the correct file. If the library file is corrupt, rebuild it using LBR. (See Chapter 3 for details.)

```
MAC -- I/O ERROR ON INPUT FILE
```

Description

The file system detected an error while reading a record from a source input file or MACRO library file. For example, it found a line containing more than 132 characters. This message can also indicate a problem with a device, a corrupt source file, or a corrupt MACRO library.

Suggested Response

Make sure all input lines do not exceed 132 characters. Check input file specifications for errors.

```
MAC -- I/O ERROR ON MACRO LIBRARY FILE
```

Description

This message has the same meaning as I/O ERROR ON INPUT FILE, except that the file is a MACRO library file and not a source input file.

Suggested Response

Make sure the library file has the correct format and is not corrupt. Check all library file specifications.

```
MAC -- I/O ERROR ON OUTPUT FILE
```

Description

The file system detected an error in writing a record to the object output file or the listing output file. This message can also indicate that a device problem exists or the device is full.

Suggested Response

Ensure that there is enough space on the output media, the output media is on line and ready, and the device is not write-locked.

```
MAC -- I/O ERROR ON WORK FILE
```

Description

A read or write error occurred on the work file used to store the symbol table. A read or write error occurs when there is a hardware problem on a device or an attempt to write to a device that is full.

Suggested Response

Ensure that there is enough space on the output media, the output media is on line and ready, and the device is not write-locked.

```
MAC -- OPEN FAILURE ON INPUT FILE
```

Description

One of the following conditions causes this error:

1. The specified device, directory, or file does not exist.
2. The volume is not mounted.

3. You do not have access to the file.
4. A hardware problem exists with the device.

Suggested Response

Check for each error condition. Also check for spelling errors in the input file specifications.

```
MAC -- OPEN FAILURE ON OUTPUT FILE
```

Description

One of the following conditions causes this error:

1. The specified device or directory does not exist.
2. The volume is not mounted.
3. The volume is full or the device is write-protected.
4. A hardware problem exists with the device.

Suggested Response

Check for each error condition. Also check for spelling errors in the output file specifications.

```
MAC -- 64K STORAGE LIMIT EXCEEDED
```

Description

64K words of work file memory are available to MACRO. This message indicates that the assembler has generated too many symbols (13,000 to 14,000) and it has run out of space. This means either the source program is too large or it contains a condition that leads to excessive size, such as a macro expansion that recursively calls itself without a terminating condition.

Suggested Response

Check for recursive macro expansions. Try to use fewer macros. Segment the program and assemble the separate parts, using global references.

Chapter 3
Using the Librarian Utility Program (LBR)

With the Librarian Utility Program (LBR) you can create, update, modify, list, and maintain user-generated object, macro, and universal library files. LBR files contain two directory tables: an entry point table (EPT) that contains entry point names (global symbols), and a module name table (MNT) that contains module names. Both the EPT and MNT are alphabetically ordered.

Object module names are derived from .TITLE directives, while entry point names are derived from defined global symbols. Once an entry point is located, its associated module can be directly accessed.

Macro module names are derived from .MACRO directives; macro entry point names are not applicable to library processing.

Universal module names are derived from file names at insert time; universal entry point names are not applicable. You can use a universal library to contain modules inserted from any kind of file.

Chapter 1 describes how to invoke the LBR utility. This chapter contains descriptions of:

1. Library Files
2. LBR Command Line
3. LBR Switches
4. Procedures for combining Library Functions
5. LBR Restrictions
6. LBR Error Messages

3.1 Library Files

The library file consists of a one-block (256–word) library header, an entry point table (each entry point has one entry point name four words long), and a module name table (each entry has one module name four words long). In addition, each module has an eight-word header. See Appendix B for detailed information on the formats and contents of library files.

3.2 LBR Command Line

LBR command lines have the general format:

```
outfile[,listfile]=infile1[,infile2,...,infilen]
```

For a complete description of file specifications, see Section 1.2. As an alternative to using file specifications, you can use an indirect command file, as described in Section 1.3.2.2. However, LBR does not accept nested indirect command files.

3.3 LBR Switches

LBR uses switches appended to file specifications to invoke functions. These switches are summarized in Table 3–1.

Table 3–1: LBR Switches

Option Name	Switch Mnemonic	Function
Compress	/CO	Compress a library file.
Create	/CR	Create a library file.
Delete	/DE	Delete a library module and all of its entry points.
Default	/DF	Specify the default library file type.
Delete Global	/DG	Delete a library module entry point.
Entry Point	/EP	Control (include) the entry of entry point elements in the library entry point table.
	/-EP	Do not include the entry of entry point elements in the library entry point table.
Extract	/EX	Extract (read) one or more modules from a library file and write them into the specified output file.
Insert	/IN	Insert a module.
List	/LI	List module names.
	/LE	List module names and module entry points.
	/FU	List module names and full module description.
Modify Header	/MH	Modify a universal module header.
Replace	/RP	Replace a module.
	/-RP	Do not replace a module.
Spool	/SP	Spool the listing for printing.
	/-SP	Do not spool the listing.
Selective Search	/SS	Set selective search attribute in module header.
Squeeze	/SZ	Reduce the size of macro source.
	/-SZ	Do not reduce the size of macro source.

3.3.1 Compress Switch (/CO)

The Compress switch physically deletes all logically deleted records, moves all free space to the end of the file, and makes the free space available for new library module inserts. In addition, the library table specification may be altered for the resulting library. LBR accomplishes this by creating a new file that is a compressed copy of the old library file. In this compression process, the actual data in the file is compressed; however, the physical length of the file remains unchanged. The old library file is not deleted after the new file is created (see Section 3.3.3).

The /CO switch can be appended only to the output file specification. The format for specifying the Compress switch is:

```
outfile/CO:size:ept:mnt=infile
```

where:

outfile — is the file specification for the compressed version of the input file. The default extension is .OLB if the input file is an object library, .MLB if the input file is a macro library, or .ULB if the input file is a universal library. Outfile must not have the same name as infile.

/CO — is the Compress switch.

:size — is the size of the new library file in 256–word blocks. If omitted, the default size is that of the old library file.

:ept — is the number of entry point table (EPT) entries to allocate. If the value specified is not a multiple of 64, the next highest multiple of 64 is used. If omitted, the default value is the number of EPT entries in the old library file. This parameter is always set to zero for macro and universal libraries. The maximum number of entries is 4096.

:mnt — is the number of module name table (MNT) entries to allocate. If the value specified is not a multiple of 64, the next highest multiple of 64 is used. If omitted, the default value is the number of MNTs in the old library file. The maximum number of entries is 4096.

infile — specifies the library file to be compressed. The default file type is .OLB for object libraries, .MLB for macro libraries, and .ULB for universal libraries. The default file type is determined by the current default file type.

For example:

```
LBR>LIBFIL/CO:100.:156.:70.=FILE1.OLB ⏎
```

File FILE1.OLB is compressed, and a new file, LIBFIL.OLB, is created with the following attributes:

```
size  = 100 blocks
ept   = 192 entry points
mnt   = 128 module names
```

NOTE

The numbers for block size, entry points, and module names include decimal points; if omitted, the numbers are interpreted as octal values. All examples and discussions in this chapter assume decimal numbers.

3.3.2 Create Switch (/CR)

The Create switch allocates a contiguous library file on a direct access device such as a disk. It initializes the library file header, the entry point table, and the module name table. The /CR switch can be appended only to the output file specifier. The format for specifying the Create switch is:

```
outfile/CR:size:ept:mnt:libtype:infiletype
```

where:

outfile — is the file specification for the library file being created. The default file extension for libraries being created is .OLB for an object library, .MLB for a macro library, or .ULB for a universal library.

/CR — is the Create switch.

:size — is the size of the library file in 256–word blocks. The default size is 100 blocks.

:ept — is the number of entry point table (EPT) entries to allocate. The default value is 512 for object libraries. This parameter is always forced to 0 for macro libraries and universal libraries. The maximum number of entries is 4096. Once a value is specified or defaulted, an error occurs if an Insert or Replace operation exceeds the value.

:mnt — is the number of module name table (MNT) entries to allocate. The default value is 256. The maximum number of entries is 4096. Once a value is specified or defaulted, an error occurs if an Insert operation exceeds the value.

:libtype — specifies the type of library to be created. Acceptable values are OBJ for object libraries, MAC for macro libraries, and UNI for universal libraries. The default is the last value specified or implied with the /DF switch (see Section 3.3.4), or OBJ if /DF has not been specified.

:infiletype specifies the default input file type for the created universal library. If this value is not specified, the default input file type for universal libraries is .UNI. This value is not defined for object or macro libraries.

In the example below, :ept and :libtype are assigned default values, while :size has the value 50 and :mnt has the value 160:

```
LBR>LIBFIL/CR:50::160 (RET)
```

If the values you specify are not multiples of 64, the EPT and MNT are automatically expanded to the next disk block boundary. For example:

```
LBR>LIBFIL/CR::128.:64.:OBJ=FILE1,FILE2,FILE3 (RET)
```

In this example, LBR performs two functions. First, LBR creates the library file LIBFIL.OLB in the user's account on the public structure (SY:). LIBFIL has the following attributes:

:size = 100 blocks (default size)
:ept = 128 entry points
:mnt = 64 module names
:type = OBJ

Secondly, LBR inserts object modules into LIBFIL from the input files FILE1.OBJ, FILE2.OBJ, and FILE3.OBJ, which reside in the user's account on the public structure (SY:). The Insert switch is the default switch for input files (see Section 3.3.8).

3.3.3 Delete Switch (/DE)

The Delete switch logically deletes library modules and their associated entry points (global symbols) from a library file. Up to 15 library modules and their associated entry points can be deleted with one Delete switch.

When LBR begins processing the /DE switch, it displays the following message at the user terminal:

```
MODULES DELETED:
```

As modules are logically deleted from the library file, the module name is displayed at the user terminal.

If a specified library module is not contained in the library file, a message is displayed, and the processing of the current command is terminated. This message is:

```
LBR -- *FATAL* - NO MODULE NAMED "name"
```

The /DE switch can be appended only to the library file specification.

NOTE

When LBR deletes a module from a library file, the module is not physically removed from the file but is marked for deletion. This means that, although the module is no longer accessible, the file space that the module occupied is not available for use, unless the deleted module is the last module inserted. To physically remove the module from the file and make the freed space available for use, you must use the /CO switch to compress the library (see Section 3.3.1).

The form for specifying the Delete switch is:

```
outfile/DE:module1[:module2:...:modulen]
```

where:

outfile is the file specification for the library file.

/DE is the Delete switch.

:module is the name of the module to be deleted.

For example:

```
LBR>LIBFIL/DE:MOD1:MOD2:MOD3 (RET)
MODULES DELETED:
MOD1
MOD2
MOD3
```

In this example, LBR deletes the modules MOD1, MOD2, and MOD3 and their associated entry points from the library file SY:LIBFIL.OLB.

3.3.4 Default Switch (/DF)

The Default switch specifies the default library file extension. Acceptable values are .OBJ for object libraries, .MAC for macro libraries, and .UNI for universal libraries. When /DF is specified without an argument, the default value of arg is .OBJ.

Specifying a default value:

1. Sets the default extension argument for the Create switch (/CR).
2. Provides an extension default value of .MLB for macro libraries, or .ULB if a universal library is being created, or .OLB for object libraries when opening an output (library) file, except in the cases of /CO and /CR. When /CO is specified, the default applies to the library input file. When /CR is specified, the default extension is .OLB if an object library is being created, .MLB if a macro library is being created, or .ULB if a universal library is being created. The /DF switch affects only the name of the file to be opened; thereafter, the library header record information is used to determine the type of library file being processed.

The /DF switch can be issued alone or appended to a library file specification. The form for specifying the Default switch is:

```
outfile/DF:libtype...
```

or

```
/DF:libtype
```

where:

outfile — is the file specification for the library file.

/DF — is the Default switch.

libtype — is .OBJ for object library files, .MAC for macro library files, and .UNI for universal files.

When you specify an extension other than .OBJ, .MAC, or .ULB, the current default library extension is set to object libraries, and the following message is displayed:

```
LBR -- INVALID LIBRARY TYPE SPECIFIED
```

Examples:

1.
```
LBR>/DF:MAC RET
LBR>LIBFIL=INFILE RET
```

   File LIBFIL.MLB is opened for insertion.

2.
```
LBR>/DF:MAC RET
LBR>LIBFIL/DF:OBJ=INFILE RET
```

   File LIBFIL.OLB is opened for insertion.

3.
```
LBR>/DF:MAC RET
LBR>LIBFIL/CR RET
```

   Macro library LIBFIL.MLB is created.

4.
```
LBR>/DF:MAC RET
LBR>LIBFIL/CR::::OBJ RET
```

   Object library LIBFIL.OLB is created.

5.
```
LBR>/DF RET
LBR>TEMP/CO=LIBFIL RET
```

   LIBFIL.OLB is opened for compression. If LIBFIL.OLB is an object library, the file TEMP.OLB is created to receive the compressed output. If LIBFIL.OLB is a macro library (a nonstandard use of the extension .OLB), the file TEMP.MLB is created.

6.
```
LBR>/DF:OBJ RET
LBR>TEMP/CO=LIBFIL.MLB RET
```

   Assuming that file LIBFIL.MLB is a macro library, the macro library file TEMP.MLB is created to receive the compressed output.

3.3.5 Delete Global Switch (/DG)

The Delete Global switch deletes a specified entry point (global symbol) from the EPT. Up to 15 entry points may be deleted with one Delete Global switch. This switch does not affect the object module, which contains the actual symbol definition.

When LBR begins processing the /DG switch, it displays the following message on the user terminal:

```
ENTRY POINTS DELETED:
```

As entry points are deleted from the library file, each deleted entry point is displayed on the user terminal. If a specified entry point is not contained in the EPT, an error message is displayed on the user terminal, and the processing of the current command is terminated:

```
LBR -- *FATAL* - NO ENTRY POINT NAMED "name"
```

The /DG switch can be appended only to the library file specification.

The format for specifying the Delete Global switch is:

```
outfile/DG:global1[:global2:...:globaln]
```

where:

outfile is the library file specification.

/DG is the Delete Global switch.

global is the name of the entry point to be deleted.

For example:

```
LBR>LIBFIL/DG:GLOB1:GLOB2:GLOB3 (RET)
ENTRY POINTS DELETED:
GLOB1
GLOB2
GLOB3
```

In this example, the entry points GLOB1, GLOB2, and GLOB3 are deleted from the library file named SY:LIBFIL.OLB.

3.3.6 Entry Point Switch (/EP)

The Entry Point switch includes or excludes entry point elements in a library entry point table. This switch can be specified in three ways:

/EP	Include entry points in the entry point table.
/-EP	Do not include entry points in the entry point table.
/NOEP	Do not include entry points in the entry point table.

/EP causes all entry points in a module or modules to be entered in the library entry point table.

/-EP or /NOEP provides for a module to be included in a library while excluding the entry points in that module from being entered in the library entry point table.

/EP and /-EP can be applied in the same command line. For example, a particular input file with /-EP overrides the effect of /EP in the output file. /EP is the LBR default; if the switch is not specified, all entry points are entered into the library entry point table. The Entry Point switch has no effect on macro or universal libraries. The formats for specifying the Entry Point switch are:

```
outfile[/EP  ]=infile,...,infilen
       [/-EP ]
       [/NOEP]
```

or

```
outfile=infile[/EP  ],...,infilen[/EP  ]
               [/-EP ]            [/-EP ]
               [/NOEP]            [/NOEP]
```

or

```
outfile[/EP  ]=infile,...,infilen[/EP  ]
       [/-EP ]                   [/-EP ]
       [/NOEP]                   [/NOEP]
```

where:

outfile is the output file specification. When the entry point switch is applied to this file specification, LBR assumes each of the input files contains modules for which entry points are to be either included or excluded.

infile is an input file specification. When the Entry Point switch is applied to an input file specification, LBR assumes only the input file(s) has the entry point to be included or excluded.

/–EP is useful for including modules that contain duplicate entry point names in the same library. /–EP lets you enter a module in the library without including its entry points in the library entry point table.

/–EP is also useful in the case where the Task Builder uses only module names to search for modules in an object module library. In this case, entries in the library entry point table are not required. /–EP can be used to exclude entry points in the library entry point table.

Depending on whether the Entry Point switch is applied to the output specifier or to an input specifier, it has either a global or a local effect.

When applied to the output file specifier, the Entry Point switch has a global effect. That is, LBR either includes all entry points in the entry point table or excludes all entry points from the entry point table.

When applied to an input file specifier, the Entry Point switch has a local effect. That is, LBR either includes entry points in the entry point table or excludes entries from the entry point table for only those modules to which the switch is applied.

The positive and negative forms of the switch may be applied to both the output and input file specifiers. For example, the effect of /EP applied to the output file can be overridden by applying /–EP to a specific input file.

Entry points in an object module are not affected by the Entry Point switch; the Entry Point switch permits you to either include or exclude entries in the library entry point table.

3.3.7 Extract Switch (/EX)

The Extract switch reads one or more modules from a library file and writes them into a specified output file. If more than one module is extracted, the modules are concatenated in the output file. The extract operation has no effect on the library file from which the modules are read; that file remains intact. Up to eight modules may be specified in one extract operation for object and macro libraries; however, only one module may be specified in one extract operation for a universal library.

For object and macro libraries, if no modules are specified in the command line, all modules in the library are extracted and concatenated in the output file in alphabetical order.

For universal libraries, only sequential files can be extracted to a record-oriented device such as a terminal.

The /EX switch may be applied only to input file specifications. The format for specifying /EX is:

```
outfile=infile/EX[:modulename:...modulename]
```

where:

outfile	is the file specification for the file into which extracted modules are to be stored. If the input modules are object modules, the default extension for this file is .OBJ. If the input modules are macro definitions, the default extension is .MAC. If the library is a universal library, the outfile retains the infile type of the module extracted. (However, you are allowed to extract only one universal library module at a time.)
infile	specifies the library file from which the modules are to be extracted. The default extension for this file is .ULB, .OLB or .MLB, depending on the current default library type.
/EX	is the Extract switch.
modulename	is the name of the module to be extracted from the library.

Consider the following examples:

```
LBR>DRIVER=LIBRY/EX:DXDRV:DKDRV:TTDRV (RET)
```

The object modules DXDRV, DKDRV, and TTDRV are concatenated and written into the file DRIVER.OBJ.

```
LBR>KB:=LB:TSTMAC,SML/EX:QIO$$ (RET)
```

The macro QIO$$ is displayed at the issuing terminal.

```
LBR>TEST.OBS=TEST/EX (RET)
```

All of the modules in the library TEST.OLB are written into the file TEST.OBS in alphabetical order.

3.3.8 Insert Switch (/IN) for Object and Macro Libraries

The Insert switch inserts library modules into an existing library file. An LBR command line is limited to 80 characters. Each file specified can contain any number of concatenated input modules. For macro libraries with nested macros, only first-level macro definitions are extracted from the input files. All text outside the first-level macro definitions is ignored. The /IN switch is the default library file option and can be appended only to the library file specification. Note that the number of MNTs and EPTs inserted cannot exceed the number defined for the file at its creation.

When you attempt to insert an input module that already exists in the library file, the following message is displayed on the initiating terminal:

```
?LBR -- *FATAL* DUPLICATE MODULE NAME "name" IN filename
```

Similarly, if you try to insert a module containing an entry point that already exists in the EPT, the following message is displayed on the initiating terminal:

```
?LBR -- *FATAL* DUPLICATE ENTRY POINT "name" IN filename
```

The format for specifying the Insert switch is:

```
outfile[/IN]=infile1[,infile2,...,infilen]
```

where:

outfile — is the file specification for the library file into which the input modules are to be inserted. The default extension depends on the current default (see Section 3.3.4). This extension is .OLB if the current default is object libraries and .MLB if the current default is macro libraries.

/IN — is the Insert switch.

infile is the file specification for the input file containing modules to be inserted into the library file. The default extension is .OBJ if outfile is an object library and .MAC if outfile is a macro library.

For example:

```
LBR>LIBFIL/IN=FILE1,FILE2,FILE3 ⏎
```

The modules contained in the files FILE1, FILE2 and FILE3, which reside in your account on the public structure (SY:), are inserted into the library file LIBFIL, which also resides in your account on SY:. The default extension for files FILE1, FILE2, and FILE3 is .OBJ if LIBFIL is an object module library and .MAC if LIBFIL is a macro library.

3.3.9 Insert Switch (/IN) for Universal Libraries

The Insert switch works in basically the same way for universal libraries as it does for object libraries and macro libraries. However, when inserting a file into a universal library, the /IN switch is normally applied to the input file. Furthermore, you can specify a module name and descriptive information as switch values in the command line. In addition, LBR copies input file attributes to the module header.

The Insert switch format for universal libraries is:

```
outfile=infile/IN:name:op:op:...
```

where:

outfile specifies the universal library into which the file infile is to be inserted.

infile specifies the input file to be inserted into outfile. The default for the file type is the value indicated at the universal library's creation time. (See Section 3.3.2.)

/IN specifies the Insert switch.

:name optionally specifies the module name (up to six Radix–50 characters). The default is the first six characters of the input file name.

:op specifies optional descriptive information (up to six Radix–50 characters) to be stored in the module header. If you define one or more of the options, you must include colons to hold the place for each of the preceding options in the specification.

For example:

```
LBR>RICKLB.ULB=JOE.TXT/IN:MOD1:THIS:IS:JAN2:TEXT
```

In this example, LBR inserts JOE.TXT into the universal library RICKLB.ULB as MOD1. "THIS", "IS", "JAN2", and "TEXT" are stored in the module header.

You can insert JOE.TXT without the Insert switch and its values. As a result, all the information that you normally specify with switch values assumes the defaults described in this section.

3.3.10 List Switches (/LI, /LE, /FU)

The List switches produce a printed listing of the contents of a library file. Three switches allow you to select the type of listing desired:

/LI Produces a listing of the names of all modules in the library file.

/LE Produces a listing of the names of all modules in the library file and their corresponding entry points.

/FU Produces a listing of the names of all modules in the library file and give a full module description for each; that is, size, date of insertion, and module-dependent information.

These switches can be appended only to the output file specification or the list file specification.

The /LI switch is the default value and need not be specified when a listing file has been specified or when /LE or /FU is included in the command.

The format for specifying List switches is:

```
infile[,listfile]/switch[es]
```

where:

infile is the file specification for the library file whose content is to be listed.

listfile is the optional listing file specification. If not specified, the listing is displayed at the user terminal.

switch[es] is the list option or options selected.

NOTE

If listfile is specified, its default device and account (PPN) is the same as the library file. Specify SY: if the listfile is on the public structure, and specify your own account for listfile if the library file is not on your account.

For example:

1. `LBR>LIBFIL/LI` RET

   In this example, a listing of the names of all the modules contained in file SY:LIBFIL.OLB is displayed on the user terminal.

2. `LBR>LIBFIL/LE` RET

   In this example, a listing of the names of all the modules and their entry points (contained in file SY:LIBFIL.OLB) is displayed on the user terminal.

3. `LBR>LIBFIL/FU` ⏎

   In this example, a listing of the names of all the modules and a full description of each module contained in file SY:LIBFIL.OLB are displayed on the user terminal.

4. `LBR>DK1:[200,200]LIBFIL,LP:/LE/FU` ⏎

   In this example, a listing of the names of all the modules, their entry points, and a full description of each module for file LIBFIL, residing in directory [200,200] on DK1:, is printed on the line printer.

3.3.11 Modify Header Switch (/MH)

The Modify Header switch pertains only to universal libraries and allows the user to modify the optional user-specified information in the module header.

The format of the switch is:

```
outfile/MH:module:op:op...
```

where:

outfile — specifies an output file for the universal library. The file type defaults to .ULB.

/MH — specifies the Modify Header switch.

:module — specifies the name of the module whose descriptive information is to be modified.

:op — specifies the optional user information (up to six Radix–50 characters) to be stored in the module header. The default is null and indicates that the corresponding information field is not to be changed. Entering a pound sign (#) clears the corresponding information field.

For example, the optional descriptive information for module A of RICKLB.ULB is:

```
"CAROL" "BOB" "LONI" "ALICE" "PHRED"
```

The LBR command is:

```
LBR>RICKLB/MH:A:BOB:CAROL:TED::#⏎
```

The optional descriptive information for Module A in file RICKLB is changed to:

```
"BOB" "CAROL" "TED" "ALICE" "      "
```

3.3.12 Replace Switch (/RP) for Object and Macro Libraries

The Replace switch replaces modules in an existing library file with input modules of the same name. Note that the number of EPTs placed into the file cannot exceed the number defined for the file at its creation. In addition, each input file can contain any number of concatenated input modules.

For macro libraries, only first-level macro definitions are extracted from the replacement files. LBR recognizes only uppercase characters in macro directives.

When a match occurs on a module name, the existing module is marked for deletion, and all of its entries are removed from the EPT. If there is also an entry point name match, the condition is fatal and terminates the current command with an error message (see Section 3.6.2).

As each module in the library file is replaced, a message is displayed on the user terminal. This message contains the name of the module being replaced:

```
MODULE "name" REPLACED
```

If the module to be replaced does not exist in the library file, LBR assumes that the input module is to be inserted and automatically inserts it without displaying a message.

The /RP switch can be specified in either of the following ways:

1. Global - The /RP switch is appended to the library file specification, and all of the input files are assumed to contain modules to be replaced.
2. Local - The /RP switch is appended to an input file specification, and only the file to which the /RP switch is appended is considered to contain modules to be replaced.

Global Format:

```
outfile/RP=infile1[,infile2,...,infilen]
```

where:

outfile — is the file specification for the library file. The default extension depends on the current default (see Section 3.3.4). If the current default is object libraries, the extension is .OLB, and if the current default is macro libraries, the extension is .MLB.

/RP — is the Replace switch.

infile — is the input file specification for the file that contains modules to be replaced in the library file. The default type is .OBJ if outfile is an object library or .MAC if it is a macro library.

You can use this format of the /RP switch to specify a list of input files without having to append the /–RP switch to each file.

To override the global function for a particular input file that should not be replaced, append /–RP to the desired input file specifier.

Local Format:

```
outfile=infile1/RP[,infile2/RP,...,infilen/RP]
```

where:

outfile is the file specification for the library file. The local format default is the same as the global format default described above.

infile is the input file specification for the file that contains modules to be inserted or replaced in the output library file. The local format default is the same as the global format default described above.

/RP is the Replace switch and, when appended to an input file specification, constitutes the local format of the switch. This overrides the LBR default (Insert) and instructs LBR to treat the modules contained in the specified file as modules to be replaced.

The files used in the following four examples, and the modules contained within each file, are listed in Table 3–2. For the examples, the pertinent files are assumed to reside in the default directory on the default device, and the initial state of the library file is assumed to be as listed in Table 3–2.

Table 3–2: Sample Files Used in LBR Examples

	Output Library File		Input Files		
File Name	**LIBFIL.OLB**	**FILEA.OBJ**	**FILED.OBJ**	**FILEB.OBJ**	**FILEC.OBJ**
Object Modules	FILEC1 FILEC2 FILEB1 FILEB2 FILEA	FILEA	FILED1 FILEB2 FILEB3	FILEB1 FILEC2 FILEC3	FILEC1 FILED2

1. `LBR>LIBFIL/RP=FILEA,FILEB,FILEC` RET

```
MODULE "FILEA" REPLACED
MODULE "FILEB1" REPLACED
MODULE "FILEB2" REPLACED
MODULE "FILEC1" REPLACED
MODULE "FILEC2" REPLACED
```

In this example, the global format for the /RP switch is used. Object modules from the input files FILEA, FILEB, and FILEC replace modules by the same names in the library file named LIBFIL. The resulting library file is shown in Table 3–3.

Table 3–3: Output Library File After Execution of Example 1

LIBFIL.OLB
FILEC1
FILEC2
* FILEC3
FILEB1
FILEB2
* FILEB3
FILEA

*These modules did not exist on the library file prior to the execution of this example, but they did exist on the input files. LBR, therefore, assumed that they were to be inserted. Since LBR handled these modules as a normal insert, no message was printed on the input terminal.

2. `LBR>LIBFIL=FILED,FILEA/RP` RET

```
MODULE "FILEA" REPLACED
```

In this example, the local format of the /RP switch is used. The object module FILEA from file FILEA is replaced in the library file LIBFIL. The object modules in the file FILED are inserted in the library file. (See Section 3.4.8.) The resulting library file is shown in Table 3–4.

Table 3–4: Output Library File After Execution of Example 2

LIBFIL.OLB
** FILED1
** FILED2
FILEC1
FILEC2
FILEB1
FILEB2
* FILEA

*This module replaced.

**These modules inserted.

3. `LBR>LIBFIL/RP=FILEA,FILEB,FILEC,FILED/-RP` RET

```
MODULE "FILEA" REPLACED
MODULE "FILEB1" REPLACED
MODULE "FILEB2" REPLACED
MODULE "FILEC1" REPLACED
MODULE "FILEC2" REPLACED
```

In this example, the /-RP switch overrides the global format of the command. Object modules in files FILEA, FILEB, and FILEC are processed as modules to be replaced, and file FILED is processed as a file that contains modules to be inserted. The resulting library file is shown in Table 3–5.

Table 3–5: Output Library File After Execution of Example 3

LIBFIL.OLB
** FILED1
** FILED2
FILEC1
FILEC2
* FILEC3
FILEB1
FILEB2
* FILEB3
FILEA

*These modules were inserted by default.

**These modules were specified to be inserted. Had a module of the same name been present, a fatal error message would have been issued. See Example 4.

4.
```
LBR>LIBFIL /RP=FILEA,FILEB /-RP,FILEC ↵

MODULE "FILEA" REPLACED
?LBR -- *FATAL* -- DUPLICATE MODULE "FILEB1" IN FILEB.OBJ
```

In this example, only module FILEA from file FILEA was replaced. The user specified that the modules in file FILEB not be replaced (/-RP), but inserted. One of the modules contained in file FILEB duplicated an already existing module in file LIBFIL (see Table 3–2). Therefore, LBR issued the fatal error message and terminated the processing of the current command.

3.3.13 Replace Switch (/RP) for Universal Libraries

Use the Replace switch for universal libraries in the same way as for macro and object libraries. In addition, you can specify the same values for the Replace switch as for the Insert switch for universal libraries. (See Section 3.3.9.) You can specify the /RP switch with either the infile or the outfile.

The global Replace switch format for universal libraries is:

```
outfile/RP:name:op:op,...=infile[,infile2,...,infilen]
```

The local Replace switch format for universal libraries is:

```
outfile=infile/RP:name:op:op...[,infile2,...,infilen]
```

where:

outfile specifies the universal library file.

infile specifies the input file that replaces modules in the library file. The default for the file extension is the value indicated at the universal library's creation time. (See Section 3.3.2.)

/RP specifies the Replace switch.

:name optionally specifies the module name to be replaced (up to six Radix–50 characters). The default is the first six characters of the infile name.

:op specifies optional descriptive information (up to six Radix–50 characters) to be stored in the module header. The default is null. If only part of the information set is specified, all preceding colons must be supplied.

For example:

```
LBR>TEXT.ULB=DEBBIE.TXT/RP::THIS:IS:JAN3:UPDATE
MODULE "DEBBIE" REPLACED
```

In this example, LBR replaces the DEBBIE module in the universal library TEXT.ULB with an updated module from file DEBBIE.TXT. The date of replacement is specified by the user optional information and inserted in the module header. Note that the optional name is omitted.

The initial state of the library file is shown in Table 3–6. The resulting library file is shown in Table 3–7.

Table 3 – 6: Sample Files for Universal Library Replace Example

	Output Library File	Input Files
File Name	TEXT.ULB;1	DEBBIE.TXT
Modules	DEBBIE BERNIE	

Table 3–7: Output Library File After Execution of Universal Library Replace Example

TEXT.ULB;1	
DEBBIE BERNIE	*

*The module DEBBIE was replaced. If a different infile were specified, that file would be become module DEBBIE and occupy the same location in TEXT.ULB.

3.3.14 Selective Search Switch (/SS)

The Selective Search switch sets the selective search attribute bit in the module header of each object module inserted into an object library. The switch has no effect when applied to modules being inserted into a macro library. You use the switch only on input files for insertion or replacement operations, and it affects all modules in the input file to which it is applied.

Object modules with the selective search attribute bit set are given special treatment by the Task Builder. Global symbols defined in object modules with the selective search attribute are included in the Task Builder's symbol table only if they are previously referenced by other modules. Therefore, only referenced global symbols are listed with the module in the Task Builder's memory allocation file, thereby reducing task build time. The /SS switch should be applied to object files whose modules contain only absolute (not relocatable) symbol definitions. See the *RSTS/E Task Builder Reference Manual*, Appendix C, for more information.

The format for the Selective Search switch is:

```
outfile=infile1/SS[,infile2/SS,...,infilen/SS]
```

where:

outfile is the file specification for the library file.

infile is the file specification for the input file that contains modules to be selectively searched.

/SS is the Selective Search switch.

3.3.15 Spool Switch (SP)

The Spool switch determines whether or not the file is queued to the line printer. If you include the Spool switch, the file is queued and printed, but only if the spooling package is running. The default is /–SP, which means that the file is not to be printed.

The /SP switch can be appended only to the list file specifier.

The format for the Spool switch is:

```
outfile,listfile[/SP] or [/-SP]
```

where:

outfile specifies the library file.

listfile specifies the listing file.

/SP or /–SP specifies the Spool switch.

Example

```
LBR>RICKLB/DE:SHEILA,RCKLST/SP
```

In this example:

1. The module SHEILA and its associated entry points are deleted from the library file SY:RICKLB.
2. The listing of the contents of the resulting library file RICKLB is written to the list file SY:RCKLST.LST. The file is automatically printed.

3.3.16 Squeeze Switch (/SZ)

The Squeeze switch reduces the size of macro definitions by eliminating all trailing blanks and tabs, blank lines, and comments from macro library files. This switch has no effect on object libraries or universal libraries.

The /SZ switch can be specified in a global or local format.

1. Global format – The /SZ switch is appended to the library file specification, and all of the input files are assumed to contain modules to be squeezed.
2. Local format – The /SZ switch is appended to an input file specifier, and only the file to which the /SZ switch is appended is considered to contain modules to be squeezed.

Global Format

```
outfile/SZ=infile1 [,infile2,...,infilen]
```

where:

outfile is the file specification for the library file.

/SZ is the Squeeze switch.

infile is the file specification for the input file that contains modules to be squeezed before being inserted into the library file.

You can use this format of the /SZ switch to specify a list of input files without having to append the /SZ switch to each file.

To override the global squeeze function for a particular input file that is to be inserted but not squeezed, append /–SZ or /NOSZ to the desired input file specifier.

Local Format

```
outfile=infile1/SZ[,infile2/SZ,...,infilen/SZ]
```

where:

outfile is the file specification for the library file.

infile is the file specification for the file that contains modules to be squeezed before being inserted into the library file.

/SZ is the Squeeze switch.

LBR uses the following algorithm on each line to be squeezed and inserts the resulting line into the library file:

1. LBR searches the line for the rightmost semicolon (;).
2. If it finds a semicolon, LBR deletes it, along with all trailing characters in the line.
3. LBR deletes all trailing blanks and tabs in the line.
4. If the resulting line is null, nothing is transferred to the library file.

The /SZ switch scans for semicolons from right to left and deletes text from right to left until the first semicolon is encountered. Only the rightmost semicolon and the text to its right are deleted. If the line contains a semicolon embedded in meaningful (non-comment) text and you want comments squeezed, code a dummy comment for that line. The /SZ switch uses only this rightmost comment during squeeze processing.

Figure 3–1 shows the use of the LBR /SZ switch. A file containing input text to be squeezed is illustrated, along with the text actually inserted into the library file after the squeeze operation has been completed.

Figure 3–1: MACRO Listing Before and After Running LBR with /SZ Switch

BEFORE BEING SQUEEZED

```
        .MACRO  MOVSTR RX,RY,?LBL

;***    - - NOTE :                                  ;
;       BOTH ARGUMENTS MUST BE REGISTERS            ;

LBL:    MOVE    (RX)+,(RY)+     ;MOVE A CHARACTER
        BNE     LBL             ;CONTINUE UNTIL NULL SEEN
        DEC     RY              ;BACKUP OUTPUT PTR TO
NULL

;END OR MOVSTR
        .ENDM
```

AFTER BEING SQUEEZED

```
        .MACRO  MOVSTR RX,RY,?LBL
;***    - - NOTE :
;       BOTH ARGUMENTS MUST BE REGISTERS
LBL:    MOVB    (RX)+,(RY)+
        BNE     LBL
        DEC     RY
        .ENDM
```

3.4 Combining Library Functions

You can request two or more library functions in the same command line. The only exceptions are that (1) /CO cannot be requested with anything else except /LI, /LE, or /FU and that (2) /CR and /DE cannot be specified in the same command line.

Functions are performed in the following order:

1. /DF
2. /CR or /CO
3. /DE
4. /DG
5. /IN, /RP, /SS, /SZ
6. /LI, /LE, /FU

For example:

```
LBR>FILE/DE:XYZ:$A,LP:/LE/FU=MODX,MODY/RP (RET)
```

In order, LBR:

1. Deletes modules XYZ and $A.
2. Inserts all modules from MODX and MODY, replacing any duplicates of modules in MODY.
3. Produces a listing of the resulting library file on the line printer with full module descriptions and all entry points.

3.5 LBR Restrictions

The following restrictions apply when using LBR:

1. Limit of 65,536 words per module.
2. Limit of 65,536 blocks per library.
3. Allocate tables to maximum anticipated size. To expand table allocations, use the /CO switch to copy the entire file.
4. Three conditions result in a fatal error when using the /IN switch to insert a module into a library:
   a. The name of the inserted module matches the name of a module already in the library. This error can be avoided by using the /RP switch to replace one module with another module of the same name.

b. The entry point name of the inserted module matches an entry point name of a module in the library. For further information, refer to Section 3.3.8.

c. The library cannot be extended because of the lack of disk space.

5. The use of wildcards, such as *.OBJ, where the * indicates all modules with extension .OBJ, is not allowed.

6. There must be enough space in the library's tables for both the modules being replaced and their replacements, since the new modules are entered before the old modules are marked for deletion.

3.6 LBR Error Messages

There are two types of LBR error messages: *diagnostic* and *fatal.*

Diagnostic error messages describe an existing condition that requires consideration but does not warrant termination of the command. When a hardware error is suspected, examine the system error log to determine the device and error type. Diagnostic messages are displayed at the user terminal in the format:

```
%LBR -- *DIAG* - message
```

Fatal error messages describe a condition that caused LBR to stop processing a command. When this occurs, LBR returns to the appropriate command level. For example, if the command is entered in response to the CCL command, that is,

```
LBR command RET
```

then LBR issues the fatal error message and exits. If, however, the command is entered in response to the LBR prompt, that is,

```
LBR>command RET
```

LBR issues the fatal error message and reprompts.

Fatal error messages are displayed at the user terminal in the format:

```
?LBR -- *FATAL* - message
```

If a fatal error occurs during the processing of an indirect command file, the command file is closed, the fatal error message and command line in error are displayed on the user terminal, and LBR returns to the appropriate command level.

3.6.1 Effect of Fatal Errors on Library Files

The status of a library file after fatal errors is:

1. In general, output errors leave the library in an indeterminate state.

2. During the deletion process directed by the /DE switch, the library is rewritten prior to the display of the individual module-entry-point-deleted messages.
3. During the replacement process directed by the /RP switch, the library is rewritten prior to the display of the individual module-replaced messages.
4. During the insert process directed by the /IN switch, the library is rewritten after the insertion of all modules in each individual input file.

3.6.2 List of LBR Errors

The following list of LBR error messages provides a description of the error cause along with suggested user responses.

```
LBR -- BAD LIBRARY HEADER
```

Description: Either the file is not a library file or the file is corrupt.

Suggested User Response:

1. If the file is not a library file, reenter the command line with a proper library file specified.
2. If the volume is corrupt, it must be reconstructed before it can be used.

```
LBR -- CANNOT MODIFY HEADER
```

Description: An attempt was made to modify the module header of a module in an object library or macro library. No change is made to the module header.

Suggested User Reponse: Reenter the command line, specifying a module in a universal library.

```
LBR -- COMMAND I/O ERROR
```

Description: One of the following conditions may exist:

1. A problem with the physical device (for example, device hung).
2. The file is corrupt or the format is incorrect (for example, record length exceeds 132 bytes).

Suggested User Response: Reenter the command line, using the correct syntax.

```
LBR -- COMMAND SYNTAX ERROR
command line
```

Description: A command was entered in a format not conforming to syntax rules.

Suggested User Response: Reenter the command line, using the correct syntax.

```
LBR -- DUPLICATE ENTRY POINT NAME "name" IN filename
```

Description: An attempt was made to insert a module into a library file when the insert module and a module in the library file have identical entry point names.

Suggested User Response: Determine if the specified input file is the correct file. If not, reenter the command line, specifying the correct input file. If the input file is the correct file, you can delete the duplicate entry point from the library and try again.

```
LBR -- DUPLICATE MODULE NAME "name" IN filename
```

Description: An attempt has been made to insert a module into a library that already contains a module with the specified name, without use of the /RP switch.

Suggested User Response: Determine if the specified input file is the correct file. If the input file is correct, decide whether to delete the duplicate module from the library file and insert the new one, or replace the duplicate module by rerunning LBR with the /RP switch appended to the input file specification.

```
LBR -- EPT OR MNT EXCEEDED IN filename
```

Description: The EPT or MNT table limit has been reached during the execution of an Insert or Replace command.

Suggested User Response: Copy the library, increasing the table space with the /CO switch. Reenter the command line.

```
LBR -- EPT OR MNT SPACE EXCEEDED IN COMPRESS
```

Description: An EPT or MNT table size was specified for the output library file that is too small to contain the EPT or MNT entries used in the input library file.

Suggested User Response: Reenter the command line with a larger EPT or MNT table size.

```
LBR -- ERROR IN LIBRARY TABLES, FILE filename
```

Description: The library file is corrupt or is not a library file.

Suggested User Response: If the file is corrupt, no recovery is possible and the file must be reconstructed. If the file is not a library file, reenter the command line with the correct library file.

```
LBR -- EXACTLY ONE INPUT FILE MUST APPEAR WITH /CO
```

Description: No file or more than one input library file was specified in the /CO command.

Suggested User Response: Reenter the command line with only one input file.

```
LBR -- FATAL COMPRESS ERROR
```

Description: The input library file is corrupt or is not a library file.

Suggested User Response: No recovery is possible. The file in question must be reconstructed.

```
LBR -- GET TIME FAILED
```

Description: LBR failed to execute a Get Time Parameters directive. The error is caused by a system malfunction.

Suggested User Response: Reenter the command line. If the problem persists, submit a Software Performance Report along with the related console dialogue and any other pertinent information.

```
LBR -- ILLEGAL DEVICE/VOLUME
command line
```

Description: Device specifier entered is not a valid device name. A device specifier consists of two ASCII characters, followed by one or two optional digits.

Suggested User Response: Reenter the command line with the correct device syntax specified.

```
LBR -- ILLEGAL DIRECTORY
command line
```

Description: The PPN entered does not conform to syntax rules. The PPN must have the form [n,n], where n can be one to three digits.

Suggested User Response: Reenter the command line with the correct PPN syntax.

```
LBR -- ILLEGAL FILENAME
command line
```

Description: One of the following was entered:

1. A file specification containing a wildcard.
2. A file specification that neither is a filename nor has an extension.

Suggested User Response: Reenter the command line correctly.

```
LBR -- ILLEGAL GET COMMAND LINE ERROR CODE
```

Description: The system, due to an internal failure, is unable to read a command line.

Suggested User Response: Reenter the command line. If the problem persists, submit a Software Performance Report along with the related console dialogue and any other pertinent information.

```
LBR -- ILLEGAL SWITCH
command line
```

Description: A switch was not recognized or a legal switch was specified in an invalid context.

Suggested User Response: Reenter the command line with the correct switch specification.

```
LBR -- ILLEGAL SWITCH COMBINATION
```

Description: You entered switches that cannot be executed in combination. See Section 3.4.

Suggested User Response: Reenter the command line, specifying the switches in the proper sequence.

```
LBR -- INDIRECT COMMAND SYNTAX ERROR
command line
```

Description: An indirect file was specified in a format that does not conform to syntax rules.

Suggest User Response: Reenter the command line with the correct syntax.

```
LBR -- INDIRECT FILE DEPTH EXCEEDED
command line
```

Description: An attempt has been made to exceed one level of indirect command files.

Suggested User Response: Rerun the job with only one level of indirect command file.

```
LBR -- INDIRECT FILE OPEN FAILURE
command line
```

Description: The requested indirect command file does not exist. One of the following conditions may exist:

1. You tried to read a file and were denied access.
2. A problem exists on the physical device.
3. The volume is not mounted.
4. The specified file directory does not exist.
5. The specified file does not exist.

Suggested User Response: Determine which of the above conditions caused the message and correct that condition. Reenter the command line.

```
LBR -- INPUT ERROR ON filename
```

Description: The file system, while attempting to process an input file, has detected an error. A problem exists with the physical device due to some transient condition.

Suggested User Response: Reenter the command line.

```
LBR -- INSUFFICIENT DYNAMIC MEMORY TO CONTINUE
```

Description: Job swap max is too small for LBR.

Suggested User Response: Run LBR with a larger job swap max. (Refer to the *RSTS/E System Generation Manual* for more information.)

```
LBR -- INVALID EPT AND/OR MNT SPECIFICATION
```

Description: An EPT or MNT value greater than 4096 was entered in a /CR or /CO switch.

Suggested User Response: Reenter the command line with a valid value.

```
LBR -- INVALID MODULE FORMAT, insertion module
```

Description: An attempt was made to insert a macro module into an object library.

Suggested User Response: Determine if an object file should be inserted into an object library. If so, reenter the command line with the correct object file. If a macro library was to receive the insertion, reenter the command line with the correct macro library.

```
LBR -- INVALID FORMAT, INPUT FILE filename
```

Description: The format of the input file is not the standard format for a macro source or object file, or the input file is corrupt.

Suggested User Response: Reenter the command line with the correct input file.

```
LBR -- INVALID OPERATION FOR OBJECT AND MACRO LIBRARIES
```

Description: Module header information was supplied for an object library or macro library in an Insert or Replace command.

Suggested User Response: No action required. The command is executed as if the information had not been supplied.

```
LBR -- INVALID LIBRARY TYPE SPECIFIED
```

Description: An illegal library extension in a Create (/CR) or Default (/DF) command line. The extensions .OBJ and .MAC are the only valid specifications. See Sections 3.3.2 and 3.3.4.

Suggested User Response: Reenter the command line with .OBJ or .MAC specified.

```
LBR -- INVALID NAME -- "name"
```

Description: A module name or entry point that contains a character that is not in the Radix–50 character set has been specified for deletion. Radix–50 characters consist of the letters A through Z, the numbers 0 through 9, and the special characters period (.) and dollar sign ($).

Suggested User Response: Reenter the command line with a valid name.

```
LBR -- INVALID RAD50 CHARACTER IN "character string"
```

Description: A character you supplied as part of information when you used the Insert, Replace, or Modify Header switch for a universal library is not a Radix–50 character.

Suggested User Response: Determine which character of the corresponding switch value is not a Radix–50 character. Reenter a Radix–50 character in place of the invalid character.

```
LBR -- I/O ERROR INPUT FILE filename
```

Description: A read error has occurred on an input file. One of the following conditions may exist:

1. A problem exists on the physical device.
2. The file is corrupt or the format is wrong (record length exceeds 132 bytes).

Suggested User Response: Determine which of the above conditions caused the message and correct that condition. Reenter that command line.

```
LBR -- LIBRARY FILE SPECIFICATION MISSING
```

Description: A command was entered without specifying the library file.

Suggested User Response: Reenter the command line with the library file specified.

```
LBR -- MARK FOR DELETE FAILURE ON LBR WORK FILE
```

Description: When LBR begins processing commands, it automatically creates a work file marked for deletion. For some reason, this operation failed.

Suggested User Response: Reenter the command line.

```
LBR -- MULTIPLE MODULE EXTRACTIONS NOT PERMITTED FOR UNV MODULES
```

Description: An attempt was made to extract more than one module from a universal library. The first module specified is extracted but others are ignored.

Suggested User Response: Reenter the command line for each additional extraction.

```
LBR -- MISSING OUTPUT FILE SPECIFIER
```

Description: The outfile specification was not included in the LBR command line.

Suggested User Response: Reenter the command line with the outfile specification included.

```
LBR -- NO ENTRY POINT NAMED "name"
```

Description: The entry point to be deleted is not in the specified library file.

Suggested User Response: Determine if the entry point is misspelled or if the wrong library file is specified. Reenter the command line with the entry point correctly specified.

```
LBR -- NO MODULE NAMED "module"
```

Description: The module to be deleted is not in the specified library file.

Suggested User Response: Determine if the module name is misspelled or if the wrong library file is specified. Reenter the command line with the module name correctly specified.

```
LBR -- OPEN FAILURE ON FILE filename
```

Description: The file system, while attempting to open a file, has detected an error. One of the following conditions may exist:

1. You tried to read a file and were denied access.
2. A problem exists on the physical device.
3. The volume is not mounted.
4. The specified file directory does not exist.
5. The specified file does not exist.
6. There is insufficient contiguous space to allocate the library file (this applies to the Compress and Create switches only).

Suggested User Response: Determine which of the above conditions caused the message and correct that condition. Reenter that command line.

```
LBR -- OPEN FAILURE ON LBR WORK FILE
```

Description: While you attempted to open the LBR work file, an error was detected. One of the following conditions may exist:

1. The volume is full.
2. The device is write-protected.
3. A problem exists with the physical device.

Suggested User Response: Determine which of the above conditions caused the message and correct that condition. Reenter the command line.

```
LBR -- OUTPUT ERROR ON filename
```

Description: A write error has occurred on the output file. One of the following conditions may exist:

1. The volume is full.
2. The device is write-protected.
3. The hardware has failed.

Suggested User Response: If the volume is full, delete all unnecessary files and rerun LBR. If the device is write-protected, logically dismount write-enable, logically remount, then reenter the command line. If the hardware has failed, assign a new device and reenter the command line.

```
LBR -- RMS FILES CANNOT BE EXTRACTED TO RECORD ORIENTED DEVICE
```

Description: An attempt was made to extract to a record-oriented device (such as a KB: or LP:) a module inserted from a non-sequential RMS file (such as a relative or index file). This is a fatal error message.

Suggested User Response: Extract the file to a disk and then use an RMS conversion to make an RMS sequential file.

```
LBR -- TOO MANY INPUT FILES
```

Description: Too many input file specifications were included in the LBR command line. You are limited to 80 characters.

Suggested User Response: Reenter a command line not exceeding 80 characters.

```
LBR -- TOO MANY OUTPUT FILES SPECIFIED
```

Description: More than two output files were specified. LBR makes the following assumptions:

1. The first output file specified is the output library file.
2. The second output file specified is the listing file.

Suggested User Response: No action is required. LBR ignores any remaining file specifications.

```
LBR -- POSITIONING ERROR ON filename
```

Description: The device is write-locked.

Suggested User Response: If the device is write-locked, logically dismount the device, write-enable it, logically remount it, and reenter the command line.

```
LBR -- VIRTUAL STORAGE REQUIREMENTS EXCEED 65536 WORDS
```

Description: This error may occur with maximum size libraries in conjunction with a single command line that logically deletes a large number of modules and entry points, and continues to replace them with an equally large number of modules and entry points having highly dissimilar names. Normally, this message indicates some sort of internal system error.

Suggested User Response: Rerun the job, but divide the complicated command line into several smaller command lines that do the same operations.

```
LBR -- WORK FILE I/O ERROR
```

Description: A write error has occurred on the LBR work file. One of the following conditions may exist:

1. The volume is full.
2. The device is write-protected.
3. The hardware has failed.

Suggested User Response: If the volume is full, delete all unnecessary files and rerun. If the device is write-protected, logically dismount the device, write-enable it, logically remount it, and reenter the command line. If the hardware has failed, assign a new device and retry the command.

Chapter 4
Using The Object Module Patch Utility (PAT) Program

The Object Module Patch Utility Program (PAT) allows you to patch or update object code that is in a relocatable binary object module. Although PAT can in theory patch any binary object file, in practice it is feasible to use PAT to patch only .OBJ files generated by MACRO-11. PAT accepts a file containing corrections or additional instructions, and applies these corrections or additions to the original object module to produce an updated object module. Figure 4-1 illustrates this procedure. Also, PAT allows you to increase the size of object modules since changes are made before the module is linked by the Task Builder.

This chapter covers the following major topics:

- How PAT Works
- Specifying the PAT Command String
- How PAT Applies Updates
- PAT Messages

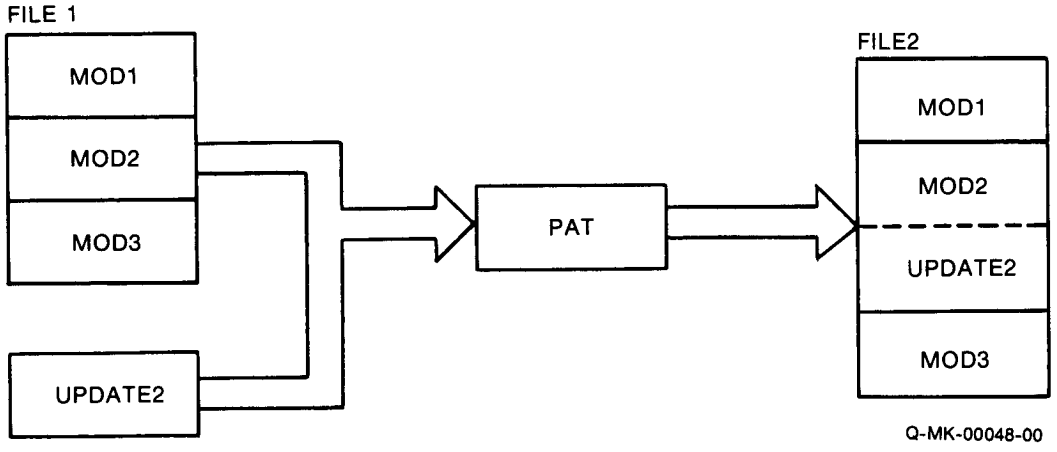
4.1 How PAT Works

PAT receives input from two files: the file being corrected and a correction file. The input file consists of one or more object modules in a single file. You may correct only one of these object modules with a single execution of PAT. The correction file consists of previously assembled object code containing corrections and additions to the input file. When linked by the Task Builder, the correction file either overlays or is added to the original object module. Output from PAT is the updated, or new object file.

You can invoke PAT using any of the methods for invoking a utility as described in Chapter 1.

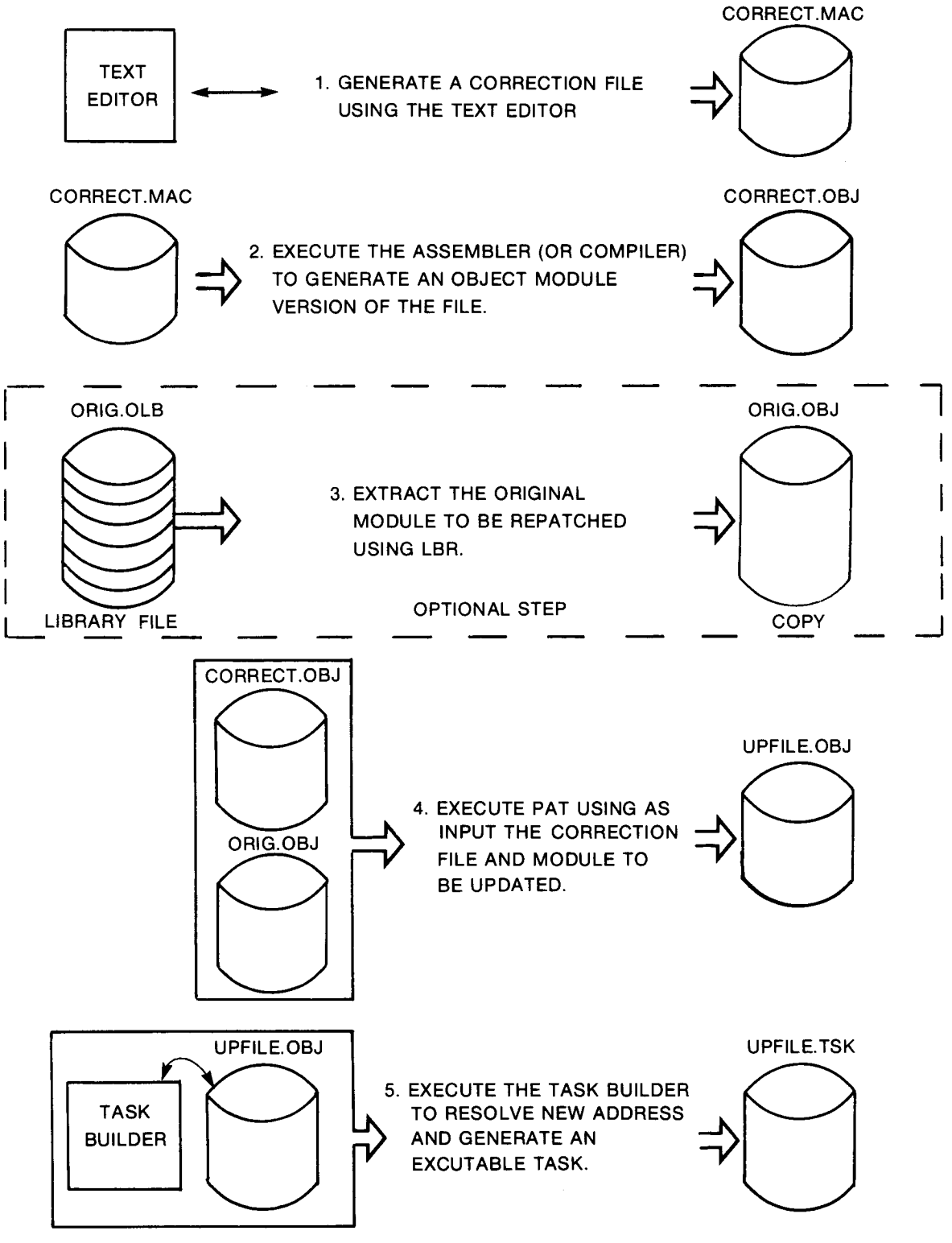
Figure 4–1 shows how PAT updates a file (FILE1) consisting of three object modules (MOD1, MOD2, and MOD3) by appending a correction file to MOD2. The updated module is then relinked with the rest of the file by the Task Builder to produce an executable task.

Figure 4–1: Updating a Module Using PAT



There are several steps involved when using PAT to update a file. First, you must create the correction file by using PIP (see *RSTS/E System User's Guide*) or a text editor. The correction file must then be assembled to produce an object module. The input file and the correction file are then submitted to PAT for processing. Finally, the updated object module, along with the object modules that comprise the rest of the file, can be submitted to the Task Builder to resolve global symbols and create an executable task. Figure 4–2 illustrates the processing steps involved in generating an updated task file using PAT.

Figure 4-2: Processing Steps Required to Update a Module Using PAT



NOTE: PREFORM STEP 3 ONLY IF THE ORIGINAL MODULE IS ON THE LIBRARY FILE.

F-MK-00047-00

4.2 Specifying the PAT Command Line

A PAT command line has the following form:

```
[outfile]=infile[/CS[:number]],correctfile[/CS[:number]]
```

where:

outfile — is the file specification for the output file. Outfile must have a different name than the infile. If you do not specify an output file, none is generated.

infile — is the file specification for the input file. This file can contain one object module or several concatenated object modules.

correctfile — is the file specification for the correction file. This file contains the corrections being applied to a single module in the input file.

/CS — specifies the Checksum switch (/CS), which directs PAT to generate an octal value for the sum of all the binary data for the module in the file to which the switch is applied. (See Section 4.3.5.)

number — specifies an octal value that PAT compares to the computed checksum value.

NOTE

PAT accepts indirect command files (see Section 1.3.2).

4.3 How PAT Applies Updates

PAT applies updates to a base input module using additions and corrections supplied in a correction file. This section describes the format of input and correction files, provides information on how to create a correction file, along with pertinent examples.

4.3.1 The Input File

The input file is the file to be updated, and therefore is the base for the updated output file produced. The input file must be in object module format. When PAT executes, the correction file module is applied to this file.

4.3.2 The Correction File

The correction file also must be in object module format. It is usually created from a MACRO-11 assembler source file in the following format:

```
.TITLE inputname
.IDENT updatenum
inputline
inputline
        *

        *

        *
```

where:

inputname — is the name of the module to be updated by PAT, and must be the same as specified in the input file's .TITLE directive for that module.

updatenum — is any value acceptable to the MACRO-11 assembler. Generally, this value identifies the update version of the file being processed by PAT, as shown in the examples below.

inputline — is each line of input to be used to correct and update the input file.

During execution, PAT adds the new global symbols defined in the correction file to the module's symbol table. Duplicate global symbols in the correction file supersede their counterparts in the input file, provided both definitions are either relocatable or absolute.

A duplicate PSECT or CSECT supersedes the previous PSECT or CSECT, provided:

1. Both have the same relocatability attribute (ABS or REL).
2. Both are defined with the same directive (.PSECT or .CSECT).

If PAT encounters duplicate PSECT names, the length attribute for the PSECT is set to the length of the longer PSECT and a new PSECT name is assigned to the other module.

If a transfer address is specified, it supersedes that of the module being patched.

4.3.3 Creating the Correction File

Referring to Figure 4–2, the first step in using PAT to update an object file is to generate the correction file. Use any editor program to generate these additions and corrections to your file.

The correction file must be in object module format before it can be processed by PAT. When you have created the source version of the correction file, you must have it assembled in order to produce an object module that PAT can process.

4.3.4 How PAT and the Task Builder Update Object Modules

The following examples show an input file and a correction file to be processed by PAT and Task Builder, along with a source-like representation of the output file after PAT and Task Builder complete processing. Programs that used the patched object module must be re-task built. Before Task-building, corrections and additions are only appended to the patched object module. After task-building, the additions and corrections are placed in their proper locations in the task image. Two techniques are used in this process; overlaying lines in a module, and appending a subroutine to a module.

4.3.4.1 Overlaying Lines in a Module — The example below illustrates a technique for overlaying lines in a module using a patch file. First, PAT appends the correction file to the input file. The Task Builder is then executed to replace code within the input file. The input file for this example is:

```
        .TITLE  ABC
        .IDENT  /01/
ABC::
        MOV     A,C             ;
        CALL    XYZ             ;
        RETURN                  ;
        .END
```

To add the instruction ADD A,B after the CALL instruction, include the following patch source file:

```
        .TITLE  ABC
        .IDENT  /01.01/
.=.+12
        ADD     A,B             ;
        RETURN                  ;
        .END
```

The patch source is assembled using MACRO-11, and the resulting object file is input to PAT along with the original object file. The updated object module appears as follows:

```
        .TITLE  ABC
        .IDENT  /01.01/
ABC::
        MOV     A,C             ;
        CALL    XYZ             ;
        RETURN                  ;
.=ABC
.=.+12
        ADD     A,B             ;
        RETURN                  ;
        .END
```

After Task Builder processes these files, the task image appears as follows:

```
        .TITLE  ABC
        .IDENT  /01.01/
ABC::
        MOV     A,C             ;
        CALL    XYZ             ;
        ADD     A,B             ;
        RETURN                  ;
        .END
```

The Task Builder uses the .=.+12 in the program counter field to determine where to begin overlaying instructions in the program. The Task Builder overlays the RETURN instruction with the patch code:

```
        ADD     A,B
        RETURN
```

4.3.4.2 Adding a Subroutine to a Module — This example illustrates a technique for adding a subroutine to an object module. In many cases, a patch requires that more than a few lines be added to patch the file. A convenient technique for adding new code is to append it to the end of the module in the form of a subroutine. This way, you can insert a CALL instruction to the subroutine at an appropriate location. The CALL directs the program to branch to the new code, execute that code, and then return to in-line processing. The input file for the example is:

```
 1                                                      .ENABL  GBL
 2
 3
 4                                                      .TITLE  ABC
 5                                                      .IDENT
 6 000000  016767  000000G 000000G ABC::   MOV     A,B     ;
 7 000006                                               CALL    XYZ     ;
   000006  004767  000000G                               JSR     PC,XYZ
 8 000012  016700  000000G                              MOV     C,R0    ;
 9 000016                                               RETURN          ;
   000016  000207                                       RTS     PC
10                                      ;              *
11                                      ;              *
12                                      ;              *
13          000001                                     .END
```

The correction file for this example is:

```
 1                                                      .ENABL  GBL
 2
 3
 4                                                      .TITLE  ABC
 5                                                      .IDENT  /01.01/
 6 000000                                               CALL    PATCH   ;
   000000  004767  000000G                               JSR     PC,PATCH
 7 000004  000240                                       NOP             ;
 8 000000                                               PSECT   PATCH   ;
 9 000000                                  PATCH:
10 000000  016767  000000G 000000G                     MOV     A,B     ;
11 000006  016700  000000G                             MOV     D,R0    ;
12 000012  006300                                      ASL     R0      ;
13 000014                                              RETURN          ;
   000014  000207                                       RTS     PC
14          000001                                     .END
```

PAT appends the correction file to the input file, as in the overlay example. The Task Builder then processes the file and the following output file is generated:

```
 1                                                  .ENABL   GBL
 2
 3
 4                                                  .TITLE   ABC
 5                                                  .IDENT   01/.01/
 6 000000                                  ABC::
 7 000000                                  CALL     PATCH    ;
   000000  004767  000000G                 JSR      PC,PATCH
 8 000004  000240                          NOP               ;
 9 000006                                  CALL     XYZ      ;
   000006  004767  000000G                 JSR      PC,XYZ
10 000012  016700  000000G                 MOV      C,R0     ;
11 000016                                  RETURN            ;
   000016  000207                          RTS      PC
12                                         ;        *
13                                         ;        *
14                                         ;        *
15 000000                                  .PSECT   PATCH
16 000000                                  PATCH:
17 000000  016767  000000G 000000G         MOV      A,B      ;
18 000006  016700  000000G                 MOV      D,R0     ;
19 000012  006300                          ASL      R0       ;
20 000014                                  RETURN            ;
   000014  000207                          RTS      PC
21         000001                          .END
```

In this example, the CALL PATCH and NOP instructions overlay the three-word MOV A,B instruction. The NOP is required because a two-word instruction replaces a three-word instruction and NOP is required to maintain word boundary alignment. The Task Builder allocates additional storage for PSECT PATCH, writes the specified code into this program section. and binds the CALL instruction to the first address in this section. Note that the MOV A,B instruction replaced by the CALL PATCH is the first instruction executed by the PATCH subroutine.

4.3.5 Determining and Validating the Contents of a File

The Checksum switch (/CS) validates the contents of a module. The Checksum switch directs PAT to compute the sum of all binary data in a file. If specified in the form /CS:number, /CS directs PAT to compute the checksum and compare that checksum to the value specified as number.

To determine the checksum of a file, enter the PAT command line with the /CS switch appended to the file specification whose checksum is being determined, for example:

```
=INFILE/CS,INFILE.PAT<RET>
```

PAT responds to this command with the message:

```
INPUT MODULE CHECKSUM IS <checksum>
```

A similiar message is generated when the checksum for the correction file is requested.

NOTE

A checksum is an octal number which is the sum of all the eight bit binary values, less carries, comprising an object module.

To validate the changes to a file, enter the Checksum switch in the form /CS:number. PAT compares the value it computes for the checksum with the value you specify as number. If the two values do not match, PAT displays the message:

```
ERROR IN FILE <filename> CHECKSUM
```

A checksum is always a nonzero value.

4.4 PAT Messages

PAT generates messages that state checksum values and messages that describe error conditions. For checksum values and nonfatal error messages, PAT prefixes messages with:

```
%PAT -- *DIAG -
```

For error messages that describe errors causing PAT to terminate, PAT uses the prefix:

```
?PAT -- *FATAL -
```

Fatal and diagnostic errors may still result in the creation of the requested output files. All output files created prior to a fatal error should be deleted; all diagnostic error output files should be examined and a decision made on whether or not to keep them.

The messages described below are grouped according to message type, as follows:

1. Information messages.
2. Command line errors.
3. Input/output errors.
4. Errors in file contents or format.
5. Internal software errors.
6. Storage allocation errors.

4.4.1 Information Messages

The following messages describe results of checksum processing.

```
CORRECTION INPUT FILE CHECKSUM IS <checksum>
```

Description: <checksum> is the module checksum displayed in response to the /CS switch appended to a correction input file specification. The value is printed in octal.

Suggested User Response: No response necessary.

```
INPUT MODULE CHECKSUM IS <checksum>
```

Description: <checksum> is the module checksum displayed in response to the /CS switch appended to an input file specification. The value is printed in octal.

Suggested User Response: No response necessary.

4.4.2 Command Line Errors

The following errors result from failure to adhere to the command line syntax rules.

```
COMMAND LINE ERROR <command line>
```

Description: The displayed command line contains an error detected by the command line processor.

Suggested User Response: Reenter the command line using the correct syntax.

```
COMMAND SYNTAX ERROR <command line>
```

Description: The command line displayed contains a syntax error.

Suggested User Response: Reenter the command line using the correct syntax.

```
ILLEGAL INDIRECT FILE SPECIFICATION <command line>
```

Description: The displayed command line contains an indirect file specification that contains one of the following errors:

- A syntax error in the file specification.
- Specification of a non-existent indirect file.

Suggested User Response: Check for file specification syntax errors. Check that the specified file is contained in the User File Directory.

```
MAXIMUM INDIRECT FILE DEPTH EXCEEDED <command line>
```

Description: The command line displayed specifies an indirect command file that exceeds the nesting level (level 2) permitted by PAT.

Suggested User Response: Reorder your files so that they do not exceed the nesting limit.

4.4.3 File Specification Errors

The following messages are caused by errors in the specification of input or output files or related file switches.

```
CORRECTION INPUT FILE MISSING <command line>
```

Description: The command line does not contain the mandatory correction file input specification.

Suggested User Response: Reenter the command line specification including the correction file.

```
ILLEGAL DEVICE/VOLUME SPECIFIED <device name>
```

Description: <device name> does not adhere to the syntax rule for specifying device or volume.

Suggested User Response: Check the rules for specifying the device and reenter the command line with the correct device specified.

```
ILLEGAL DIRECTORY SPECIFICATION <directory name>
```

Description: The directory string displayed does not adhere to the syntax rules for specifying directories.

Suggested User Response: Reenter the command line specifying the directory string in the correct syntax.

```
ILLEGAL FILE SPECIFICATION <filename>
```

Description: The filename printed does not adhere to the syntax rules for file specifications.

Suggested User Response: Reenter the command line using the correct syntax for the filename.

```
ILLEGAL SWITCH SPECIFIED <filename>
```

Description: An unrecognized switch or switch value has been appended to the filename displayed.

Suggested User Response: Check the rules for specifying the switch and reenter the command line.

```
INVALID FILE SPECIFIED <filename>
```

Description: The filename displayed:

1. References a nonexistent device.
2. References a nonexistent PPN.

Suggested User Response: Correct the device or PPN specification and reenter the command line.

```
MULTIPLE OUTPUT FILES SPECIFIED <command line>
```

Description: Only one output file specification is accepted by PAT.

Suggested User Response: Reenter the command line with only one output file specified.

```
REQUIRED INPUT FILE MISSING <command line>
```

Description: The command line does not contain the mandatory input file specification.

Suggested User Response: Reenter the command line specification input file.

```
TOO MANY INPUT FILES SPECIFIED <command line>
```

Description: The command line displayed contains too many input file specifications. PAT accepts one input and one correction file specification.

Suggested User Response: Reenter the command line specifying the correct files.

```
SYMBOL --- IS MULTIPLY DEFINED
```

Description: Multiple definitions of a symbol has occurred.

Suggested User Response: Rename one or more of the duplicate symbols, then regenerate correction file.

```
UNABLE TO FIND FILE <filename>
```

Description: The specified input or correction file could not be located.

Suggested User Response: Ensure that the file exists. Reenter the command line.

4.4.4 Input/Output Errors

The error messages listed below are caused by faults detected while performing I/O to the specified file.

```
ERROR DURING CLOSE: FILE: <filename>
```

Description: This error is most likely to occur while attempting to write the remaining data into the output file before closing it. The principal sources of error in these circumstances are:

1. Device full.
2. Device write-locked.

Suggested User Response: Perform the appropriate corrective action and reenter the command line.

```
ERROR POSITIONING FILE <filename>
```

Description: An attempt has been made to position the file beyond end-of-file.

Suggested User Response: Submit a Software Performance Report along with the related console dialogue and any other pertinent information.

```
I/O ERROR ON INPUT FILE <filename>
```

Description: An error was detected while attempting to read the specified input file. The principal cause is a device hardware error.

Suggested User Response: Reenter the command.

```
I/O ERROR ON OUTPUT FILE <filename>
```

Description: An error occurred while attempting to write into the named output file. The most likely causes are:

1. Contiguous file cannot be extended.
2. Device full.
3. Device write-locked.
4. Hardware error from device.

Suggested User Response: Perform the appropriate corrective action and reenter the command.

4.4.5 Errors in File Contents or Format

The following errors represent inconsistencies detected by PAT in the format or contents of input or correction files.

```
ERROR IN FILE <filename> CHECKSUM
```

Description: Checksum computed by PAT for the named file does not match that supplied by the user.

Suggested User Response: Ensure that the correct checksum was specified. If the checksum was correct, either the input file or the correction file was incorrect. Rerun PAT and specify the correct file.

```
FILE <filename> HAS ILLEGAL FORMAT
```

Description: The format of the named file is not compatible with the object file format accepted by the Task Builder. The principal causes are:

1. Truncated input file.
2. Input file consists of text.

Suggested User Response: Ensure that the file is a compatible object file and resubmit it for PAT processing.

```
INCOMPATIBLE REFERENCE TO GLOBAL SYMBOL <symbol name>
```

Description: A correction file contains a global symbol whose attributes do not match any of the following input file symbol attributes:

1. Definition or Reference.
2. Relocatable or Absolute.

Suggested User Response: Update the correction input file by modifying the symbol attributes. Reassemble the file and resubmit it for PAT processing.

```
INCOMPATIBLE REFERENCE TO PROGRAM SECTION <section name>
```

Description: A correction file contains a section name whose attributes do not match one of the input file section attributes:

1. Relocatable or Absolute.
2. .PSECT or .CSECT.

Suggested User Response: Update the correction file by modifying the section attribute or changing the section type. Reassemble the file and resubmit it to PAT for processing.

```
UNABLE TO LOCATE MODULE <module name>
```

Description: A module name specified in the correction input file could not be found in the file of concatenated input modules.

Suggested User Response: Update the correction file specification to include the missing module. Reenter the command line.

```
UNABLE TO OPEN FILE
```

Description: An error occurred trying to open a file to which you had no access.

Suggested User Response: Use a privileged account or change the protection code of the file you are trying to access.

4.4.6 Internal Software Error

These errors reflect internal software error conditions. If they persist, submit a Software Performance Report along with the related console dialogue and any other pertinent information.

```
ILLEGAL ERROR-SEVERITY CODE <error data>
```

Description: An error message call has been generated containing an illegal parameter.

Suggested User Action: If the error persists, submit a Software Performance Report along with the related console dialogue and any other pertinent information.

4.4.7 Storage Allocation Error

The following error message indicates that insufficient task memory was available for storing global symbol and program section data:

```
NO DYNAMIC STORAGE AVAILABLE <storage-listhead>
```

Description: Insufficient contiguous task memory was available to satisfy a request for the allocation of storage.

<Storage-listhead> is a display of the two-word dynamic storage listhead contents in octal.

Suggested User Response: None, this message is not relevent in RSTS/E. However if this message does appear, submit a Software Performance Report.

Chapter 5
Using the MAKSIL Utility Program

The MAKSIL utility program accepts as input files the generated output of the Task Builder, a task image file (extension .TSK) and a symbol table (extension .STB). Depending on how the program was originally coded and how you specify the MAKSIL utility program, MAKSIL produces a formatted output file that can be loaded into memory as a resident library (.LIB), a run-time system (.RTS), or a multi-user task.

When generating a run-time system, a new command file (.CMD) can also be generated. When generating the .LIB, .RTS, or a task image, you have the option of including the symbol table (.STB) into RSTS/E Save Image Library (SIL) format, thus allowing symbolic patching of the output file.

5.1 Creating a Run-Time System (RTS)

In order to use MAKSIL to format task builder output (task image) into a loadable run-time system, two conditions must be met.

For the first condition:

1. The starting address of the task image (the label referenced by the .END statement) must be within the lowest 1K of memory of the read-only portion of the task.
2. The highest virtual address for the task must be 177774 (octal). The word at 177774 (octal) must contain a valid, non-zero maximum job image size entry.

The second condition requires the following step:

1. Task-build the MACRO assembled run-time system code (.OBJ), then use MAKSIL to format the Task Builder output (.TSK and .STB). MAKSIL will print the following error message if the .TSK file is not aligned properly:

   ```
   TASK MUST BE EXTENDED BY xxxxxxxx BYTES
   ```

   If the file is not properly aligned, edit the command file (.CMD) to extend a "dummy" control section by the required number of bytes to align the last .PSECT at the correct boundary and rerun MAKSIL. The "edit" mode of MAKSIL can be used to automatically modify the command file (see Section 5.2).

MAKSIL may not work correctly if the Task Builder parameters are out of range. When you task build run-time system code, specify the following Task Builder options described in Table 5–1 to set the required virtual and physical address range. (See the *RSTS/E Task Builder Reference Manual.*)

Table 5–1: Task Builder Options for Virtual and Physical Address Range

Option	Description
PAR	Define virtual address base and range. PAR also implicitly specifies the largest program (low-segment) area.
STACK	The partition size is a multiple of 4K words. If the run-time system is only 3K–words for example, the STACK option would be defined as "STACK = 1024" to reserve an additional 1K–words. If this is done, the run-time system will occupy only 3K–words of physical memory when it is loaded.
EXTSCT	RSTS/E requires the task to end at virtual address 177774. The EXTSCT option extends a control section (usually .99998) so that the vector control section (.99999) ends correctly.

Table 5–2 defines the PAR and STACK options for various run-time system sizes.

Table 5–2: Task Builder PAR and STACK Options for Various Sized Run-Time Systems

Size	Options
1K – 4K	PAR = 160000:020000*
5K – 8K	PAR = 140000:040000
9K – 12K	PAR = 120000:060000
13K – 16K	PAR = 100000:100000
17K – 20K	PAR = 060000:120000
21K – 24K	PAR = 040000:140000
25K – 28K	PAR = 020000:160000
1K 5K 9K 13K 17K 21K 25K	STACK = 3072
2K 6K 10K 14K 18K 22K 26K	STACK = 2048
3K 7K 11K 15K 19K 23K 27K	STACK = 1024
4K 8K 12K 16K 20K 24K 28K	STACK = 0000

*PAR = virtual address:number of bytes

The following example shows a control file for a dummy 4K word run-time system:

```
FILE/-HD,FILE,FILE=FOOBLD/MP
PAR=FILE:160000:020000
STACK=0
EXTSCT=.99998:000000
//
```

After task building FILE.TSK and executing MAKSIL, the command file would be edited to change the PAR, STACK, and EXTSCT options to appropriate values. The task builder is then rerun to correctly build the task.

Finally, MAKSIL is rerun to build FILE.RTS.

Run-time systems are built by first specifying the size (4K,8K,etc.) and then task building so as to include as many modules resident as will fit in the partition (leaving sufficient patch space). Then, MAKSIL is run to ine the EXTSCT value. Finally, the extended task is built and converted to a run-time system.

5.2 Creating a Resident Library

MAKSIL can also produce a resident library output file. As shown in the example below, the switch /RTS is not appended to the filename entered in response to the first MAKSIL prompt. Note that the switch /DEBUG can be used if required.

```
RUN $MAKSIL (RET)

MAKSIL V70 RSTS/E Timesharing

Resident Library name? TEST (RET)

Task-built Resident Library input file <TEST.TSK>? TEST (RET)

Include symbol table (Yes/No) <Yes>? Yes (RET)

Symbol table input file <TEST.STB>? TEST (RET)

Task Image SIL output file <TEST.SIL>? (RET)

TEST built in 23 K-words, 548 symbols in the directory

TEST.TSK renamed to TEST.TSK<104>

Ready
```

5.3 Operating Instructions

Following are the keyboard operating commands and responses for MAKSIL.

Type:

```
RUN $MAKSIL (RET)
```

After displaying its header line, MAKSIL prompts, and the user answers:

```
Resident Library name? FILE/RTS (RET)
or
Resident Library name? FILE (RET)
```

Type the name of the resident library (FILE, for example) or the run-time system name (FILE/RTS, as shown above). "RTS" is required if a run-time system is to be built. The switch /RTS signals that special conditions must be met by the .TSK file before proper conversion to a run-time system Save

Image Library (SIL) format can be made. When the switch /RTS is not used, MAKSIL assumes that a Resident Library file is to be created. The switch /DEBUG can be used when creating a run-time system or a resident library file to initiate printout of internal tables during the create process.

MAKSIL then prompts:

```
Task-built Resident Library input file <FILE.TSK>? RET
```

or

```
Task-built Run-Time System input file <FILE.TSK>? RET
```

Type the name of the .TSK file, or press the RETURN key if the default name is acceptable. If a run-time system is to be built, the task is checked for correct parameters. If a resident library is requested, the next prompt is:

```
Include symbol table
```

If in Build Mode, the program checks the format of the file as a run-time system and responds with either:

```
The run-time system is correctly aligned
```

or

```
The run-time system is not aligned
```

If in "Edit Mode", to redefine task-builder parameters.

```
Edit mode (Yes/No) <Yes> ? RET
```

At this point, in running MAKSIL, you have two options; to enter the edit mode to redefine task build parameters, or the build mode to construct the run-time system. This option is presented by the following prompt:

```
Task-builder command input file <FILE.CMD>? RET
```

If the run-time system is correctly aligned, the program will exit.

The command file is edited to modify the EXTSCT, STACK and PAR options to extend the task as necessary. The program then prompts:

```
Corrected command filename <FILE.CMD>? RET
```

If you respond with the RETURN key, the old file FILE.CMD will be renamed to FILE.BAK. The program then reminds you to rebuild the task and exits:

```
Please task-build again using FILE.CMD
```

If you answered "No" to the "Edit Mode" question, the program aborts if the task is not correctly aligned. Perform the task–build. If there are no problems, the following questions are asked:

```
Include symbol table (Yes/No) <Yes>? RET
```

Typing Yes and the RETURN key or just the RETURN key, will append a symbol table (.STB) to the run-time system. The .STB file allows you to patch the .RTS via INIT or the on-line patching mechanism. If a symbol table is requested, the prompt appears:

```
Symbol table input file <FILE.STB>? RET
```

Type the name of the .LIB or .RTS file, or press the RETURN key if the default is acceptable. MAKSIL builds the run-time system or resident library (with symbol table if requested) into the output file and displays:

```
Run-Time System output file <FILE.RTS>? RET
or
Resident Library output file <FILE.LIB>? RET
or
Task Image SIL output file <FILE.SIL>? RET
```

Type the name of the .LIB or .RTS file, or press the RETURN key if the default is acceptable. (When task building, do not give the output file the same name as the input file, or else the input file could be overwritten.) MAKSIL builds the run-time system or resident library (with symbol table if requested) into the output file and displays:

```
FILE built in 4K words, 123 symbols in the directory
```

After the MAKSIL process, the task image file is renamed so that unprivileged users can access the task image with the "HISEG=" or the "LIBR=" switch when task building their programs. The output from running a multi-user task through MAKSIL is a save image library (FILE.SIL, which is executable) and a resident library (FILE.LIB, which must be added to the list of resident libraries in order to be shared by multiple users). Refer to the *RSTS/E Task Builder Manual* for more information about building multi-user tasks.

5.4 Messages

There are three types of messages that can be encountered while using MAKSIL:

1. Fatal error messages(?)
2. Diagnostic messages(%)
3. Informational messages

These three types of messages, their causes, and user responses are described in the following sections.

5.4.1 Fatal Error Messages

```
?ODD BASE OR TRANSFER ADDRESS
```

Description: The .TSK file contains an incorrect transfer address or an odd value for a base address.

Suggested User Response: Re-task build the program, and execute MAKSIL.

```
?GARBAGE WHEN CONVERTING "nnnnn" IN "command" text
```

Description: A conversion error has occurred.

Suggested User Response: Check the .CMD file, re-task build, and execute MAKSIL.

```
?COULDN'T FIND ALIGNMENT POINT
```

Description: The alignment scan could not locate the communication vector.

Suggested User Response: Check that the task build has been performed correctly.

```
?PARTITION OR STACK PARAMETER INCORRECT FOR TASK
```

Description: You are trying to extend the task too far.

Suggested User Response: Rebuild the task with correct "PAR=" and "STACK=" commands.

```
?TASK IMAGE xxxxx.TSK CANNOT BE CONVERTED TO RUN-TIME SYSTEM yyyyy.
```

Description: Same as message.

Suggested User Response: Check that the task is defined correctly. Common problems include a starting address that is not in the first 1K memory segment, a missing vector control section (.99999), or overall incorrect run-time system design.

```
?ERROR REOPENING SYMBOL TABLE
```

Description: Opening the .STB file resulted in an error after the file had once successfully been opened.

Suggested User Response: Re-execute the MAKSIL program.

```
?ERROR WHEN OPENING file.ext -- text
```

Description: An error was encountered when opening the file "file.ext" described in error message "text".

Suggested User Response: Type in correct filename in response to question.

```
?DISK FILES ONLY, PLEASE
```

Description: An attempt has been made to open a non-disk file for input or output operations.

Suggested User Response: Enter only filenames that reside on the disk in response to MAKSIL questions.

```
?ILLEGAL SYMBOL TABLE FORMAT
```

Description: The symbol table (.STB) file does not have the file attributes of either formated binary or variable length records.

Suggested User Response: Either an improper symbol table file has been specified, or the file has been corrupted. The program will build without the symbol table. Re-run the program with a valid symbol table file to include a symbol table.

```
?ERROR GETTING A .GSD ENTRY
```

Description: In processing the symbol table (.STB) file, an error occurred that prevents finding a valid symbol table entry.

Suggested User Response: Either an improper symbol table file has been specified, or the file has been corrupted.

The program will build without the symbol table. Re-run the program with a valid symbol table file to include a symbol table.

```
?LONG FORMATTED-BINARY RECORD.
```

Description: The symbol table (.STB) file contains a formatted binary record greater than 512 bytes.

Suggested User Response: Either an improper symbol table file has been specified, or the file has been corrupted. The program will build without the symbol table. Re-run the program with a valid symbol table file to include a symbol table.

```
?ILLEGAL FORMATTED-BINARY RECORD
```

Description: The symbol table (.STB) file contains a formatted record starting at an odd byte boundary.

Suggested User Response: Either an improper symbol table file has been specified, or the file has been corrupted. The program will build without the symbol table. Re-run the program with a valid symbol table file to include a symbol table.

```
?ILLEGAL VARIABLE-LENGTH RECORD
```

Description: The symbol table (.STB) file contains a variable length record which either is greater than 512 bytes in length, or starts at an odd byte boundary.

Suggested User Response: Either an improper symbol table file has been specified, or the file has been corrupted. The program will build without the symbol table. Re-run the program with a valid symbol table file to include a symbol table.

```
?ADDRESSING OUTSIDE OF TASK LIMITS
```

Description: The program tried to access beyond the calculated end of the .TSK file. The task image is incorrect.

Suggested User Response: Task build the program again and execute MAKSIL.

```
?ERROR GETTING BLOCK xx -- text
```

Description: A GET command was performed on block xx of the output file (.RTS or .LIB), which resulted in an error, as described in error message text.

Suggested User Response: Execute the MAKSIL program again.

```
?ERROR PUTTING BLOCK xx -- text
```

Description: A PUT command was performed on block xx of the output file, which resulted in an error, as described in the error message text.

Suggested User Response: Execute the MAKSIL program again.

```
?ERROR GETTING FROM xxxx.STB -- text
```

Description: An error occurred when performing a GET command from the symbol table (.STB) file, as described in error message text.

Suggested User Response: Re-execute the programs.

```
?FATAL ERROR -- text
```

Description: An unexpected error has occurred.

Suggested User Response: Send a Software Performance Report along with an appropriate listing of the error.

5.4.2 Diagnostic Messages

```
%RUN-TIME SYSTEM MAXIMUM JOB SIZE (xx) EXCEEDS CALCULATED MAXIMUM OF (yy)
```

Description: The maximum size of a particular job (O.SIZE) as defined in the .TSK, is too great for the run-time system. For example, while assembling a run-time system requiring 16K words, a job size of 28K words had been defined. Since the run-time system and a job cannot exceed 32K words, the RSTS/E Monitor adjusts the maximum job size to 16K words.

Suggested User Response: No response is required.

```
%MULTIPLE command: "first command", "command line"
```

Description: A PAR, STACK, or EXTSCT command appears more than once. Only the first command, of a particular type, is used.

Suggested User Response: No response is required.

5.4.3 Informational Messages

```
INCORRECT FILE SIZE xx, COMPUTED=yy
```

Description: The actual file size is less than that calculated from parameters contained in the .TSK file.

Suggested User Response: No response is required.

```
THE RUN-TIME SYSTEM IS NOT ALIGNED
```

or

```
THE RUN-TIME SYSTEM IS CORRECTLY ALIGNED
```

Description: One of the two messages above is displayed, depending on the outcome of the task verification phase.

Suggested User Response: No response is required.

```
THE COMMAND FILE IS ALREADY CORRECT,EXITING,
```

Description: The edit mode was selected even though the task is correct. This may happen if MAKSIL is run from a batch stream

Suggested User Response: No response is required.

```
THE TASK- BUILDER COMMANDS HAVE BEEN CHANGED AS FOLLOWS
     OLD par                          NEW par
     OLD stack                        NEW stack
     OLD extsct                       NEW extsct

<filename> will load in a xx K-words partition using yy K-words
physical memory
zz (octal) bytes may be used for expansion,
please task-build again using <filename>.CMD
```

Description: The above message is displayed to log the edit mode changes.

Suggested User Response: Re-task build using the edited command file.

```
UTILITY ADD SUPPRESSED
```

Description: This message is printed if the run-time system was not written to account [0,1].

Suggested User Response: No response is required.

Appendix A
MACRO–11 Diagnostic Error Message Summary

A diagnostic error code is printed as the first character in a source line which contains an error detected by MACRO–11. This error code identifies a syntactical problem or some other type of error condition detected during the processing of a source line. An example of such a source line is shown below:

```
Q     26 000236  010102          MOV R1,R2,A
```

The extraneous argument A in the MOV instruction above causes the line to be flagged with a Q (syntax) error.

Error Code	Meaning
A	Assembly error. Because many different types of error conditions produce this diagnostic message, all the possible directives which may yield a general assembly error have been categorized below to reflect specific classes of error conditions:

CATEGORY 1: Illegal Argument Specified.

.RADIX -- A value other than 2, 8, or 10 is specified as a new radix.

.LIST/.NLIST -- Other than a legally defined argument is specified with the directive.

.ENABL/.DSABL -- Other than a legally defined argument is specified with the directive.

.PSECT -- Other than a legally defined argument is specified with the directive.

.IF/.IIF -- Other than a legally defined conditional test or an illegal argument expression value is specified with the directive.

.MACRO -- An illegal or duplicate symbol found in dummy argument list.

CATEGORY 2: Null Argument or Symbol Specified.

.TITLE -- program name is not specified in the directive, or first non-blank character following the directive is a non-Radix-50 character.

.IRP/.IRPC -- No dummy argument is specified in the directive.

.NARG/.NCHAR/.NTYPE -- No symbol is specified in the directive.

.IF/.IIF -- No conditional argument is specified in the directive.

CATEGORY 3: Unmatched Delimiter/Illegal Argument Construction.

/ASCII/.ASCIZ/.RAD50/.IDENT -- Character string or argument string delimiters do not match, or an illegal character is used as a delimiter, or an illegal argument construction is used in the directive.

.NCHAR -- Character string delimiters do not match, or an illegal character is used as a delimiter in the directive.

CATEGORY 4: General Addressing Errors

This type of error results from one of several possible conditions:

1. Permissible range of a branch instruction. i.e., from -128 to +127 words, has been exceeded.
2. A statement makes invalid use of the current location counter, e.g., a ".=expression" statement attempts to force the current location counter to cross program section (.PSECT) boundaries.
3. A statement contains an invalid address expression. In cases where an absolute address expression is required, specifying a global symbol, a relocatable value, or a complex relocatable value results in an invalid address expression. Similarly, in cases where a relocatable address expression is required, either a relocatable or absolute value is permissible, but a global symbol or a complex relocatable value in the statement likewise results in an invalid address expression. Specific cases of this type of error are those which follow:

   .BLKB/.BLKW/.REPT -- Other than an absolute value or an expression which reduces to an absolute value has been specified with the directive.
4. Multiple expressions are not separated by a comma. This condition causes the next symbol to be evaluated as part of the current expression.

CATEGORY 5: Illegal Forward Reference.

This type of error results from either of two possible conditions:

1. A global assignment statement (symbol==expression) contains a forward reference to another symbol.
2. An expression defining the value of the current location counter contains a forward reference.

B Bounding error. Instructions or word data are being assembled at an odd address. The location counter is incremented by 1.

D Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.

E End directive not found. When the end-of-file is reached during source input and the .END directive has not yet been encountered, MACRO-11 generates this error code, ends assembly pass 1, and proceeds with assembly pass 2.

I Illegal character detected. Illegal characters which are also non-printable are replaced by a question mark (?) on the listing. The character is then ignored.

L Input line is greater than 132 characters in length. Currently, this error condition is caused only through excessive substitution of real arguments for dummy arguments during the expansion of a macro.

M Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a label previously encountered.

N A number contains a digit that is not in the current program radix. The number is evaluated as a decimal value.

O Opcode error. Directive out of context. Permissible nesting level depth for conditional assemblies has been exceeded. Attempt to expand a macro which was unidentified after .MCALL search.

P Phase error. A label's definition of value varies from one assembly pass to another or a multiple definition of a local symbol has occurred within a local symbol block. Also, when in a local symbol block defined by the .ENABL LSB directive, an attempt has occurred to define a local symbol in a program section other than that which was in effect when the block was entered. A P error code also appears if an .ERROR directive is assembled.

Q Questionable syntax. Arguments are missing, too many arguments are specified, or the instruction scan was not completed.

R Register-type error. An invalid use of or reference to a register has been made, or an attempt has been made to redefine a standard register symbol without first issuing the .DSABL REG directive.

T Truncation error. A number generated more than 16 bits in a word, or an expression generated more than 8 significant bits during the use of the .BYTE directive or trap (EMT or TRAP) instruction.

U Undefined symbol. An undefined symbol was encountered during the evaluation of an expression; such an undefined symbol is assigned a value of zero. Other possible conditions which result in this error code include unsatisfied macro names in the list of .MCALL arguments and a direct assignment (symbol=expression) statement which contains a forward reference to a symbol whose definition also contains a forward reference; also, a local symbol may have been referenced that does not exist in the current local symbol block.

Z Instruction error. The instruction so flagged is not compatible among all members of the PDP-11 family.

Appendix B
Librarian Utility Program (LBR) Files and Formats

A library file consists of a library header, an entry point table, a module name table, the library modules, and (usually) free space. The entry point table has zero length for macro and universal libraries. See Figure B–1.

B.1 Library Header

The header section is a full block (256 words) in which the first 24 words describe the current status of the library. Its contents are updated as the library is modified, so LBR can access the information it needs to perform its functions (such as Insert, Compress). The 24th word in the library header is the default insert file extension for universal libraries and is undefined for macro and object libraries. See Figure B–2.

B.2 Entry Point Table

The entry point table consists of 4–word elements that contain an entry point name (words 0–1) and a pointer to the module header where the entry point is defined (words 2–3). See Figure B–3. This table is searched when a library module is referenced by one of its entry points. The table is sequenced in order of ascending entry point names. The entry point table is not used for macro or universal library files.

B.3 Module Name Table

The module name table is searched when the library module is referenced by its module name rather than by one of its entry points. It is comprised of 4–word elements; a module name (words 0–1) and a pointer to the module header (words 2–3). See Figure B–4. The module name table is sequenced in order of ascending module names.

B.4 Module Header

Each module starts with a header of 8 words for object and macro modules and 32 words for universal modules, identifying the type and status of the module, its length (number of words), and so forth (see Figure B–5).

For object modules, the low-order bit of the attributes byte is set if the module has the selective search attribute. (See Section 3.3.14, Selective Search Switch (/SS). The selective search attribute reduces task build time.) In addition, for object modules, the two words of type-dependent information contain the module identification defined by the .IDENT directive at assembly time. For macro modules, these two fields are undefined.

For universal modules, type-dependent identification is derived from the file extension of the input file. See Figure B–7.

Universal libraries allow module header changes (optional descriptive information) by the /MH switch.

Figure B–1: Standard Library File Format

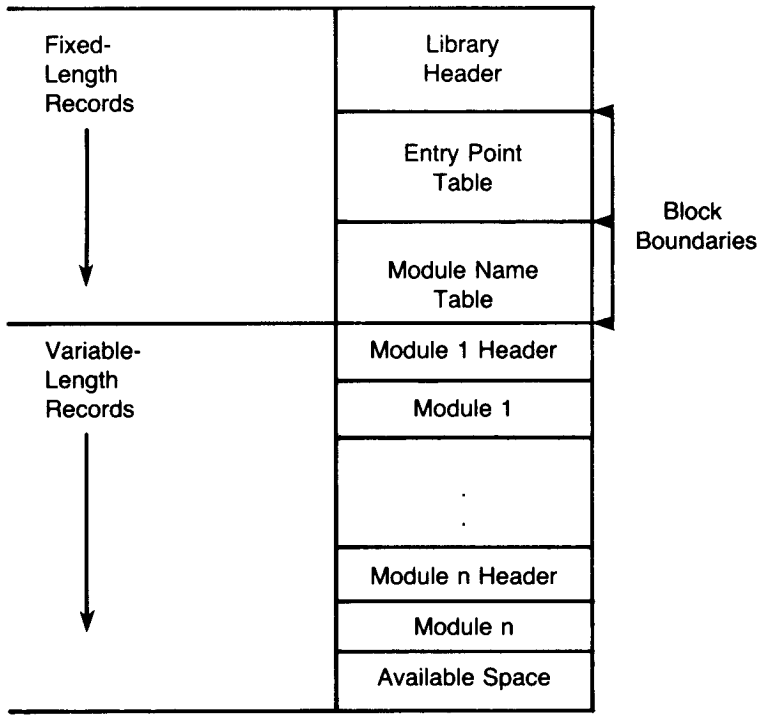
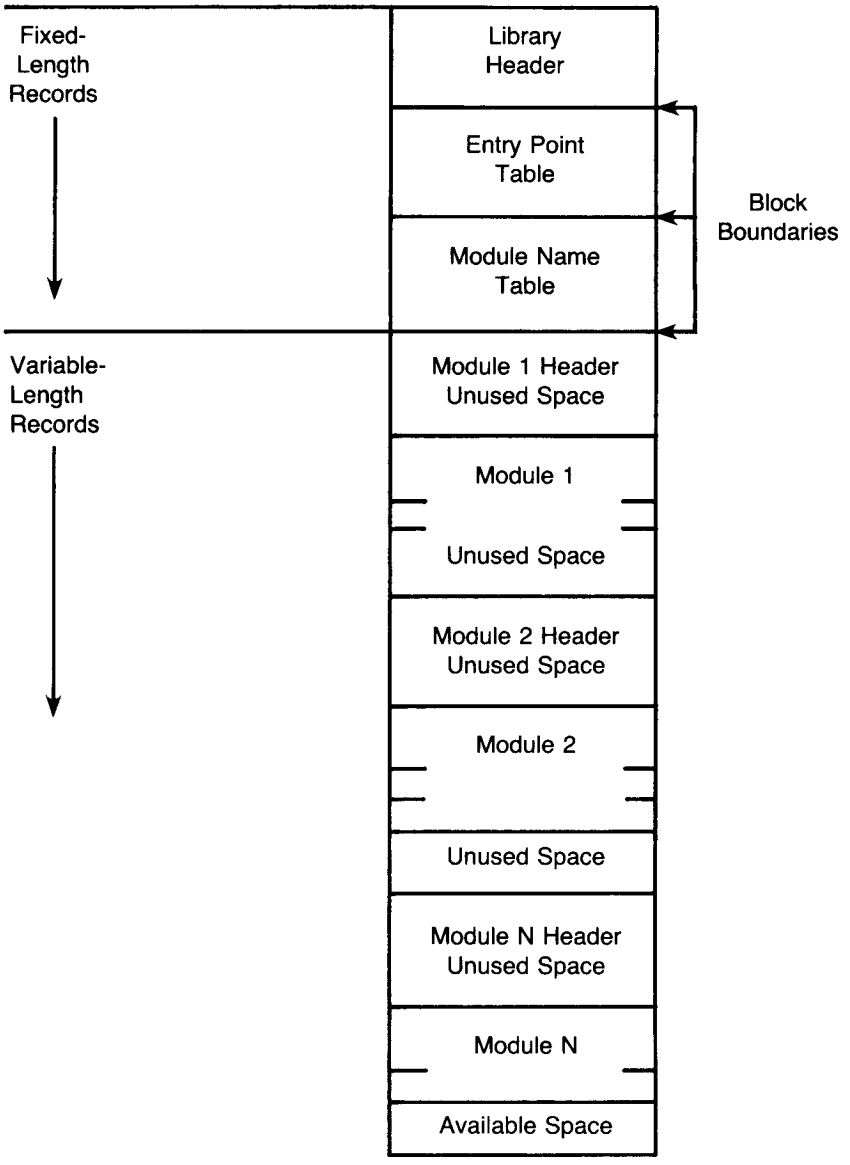


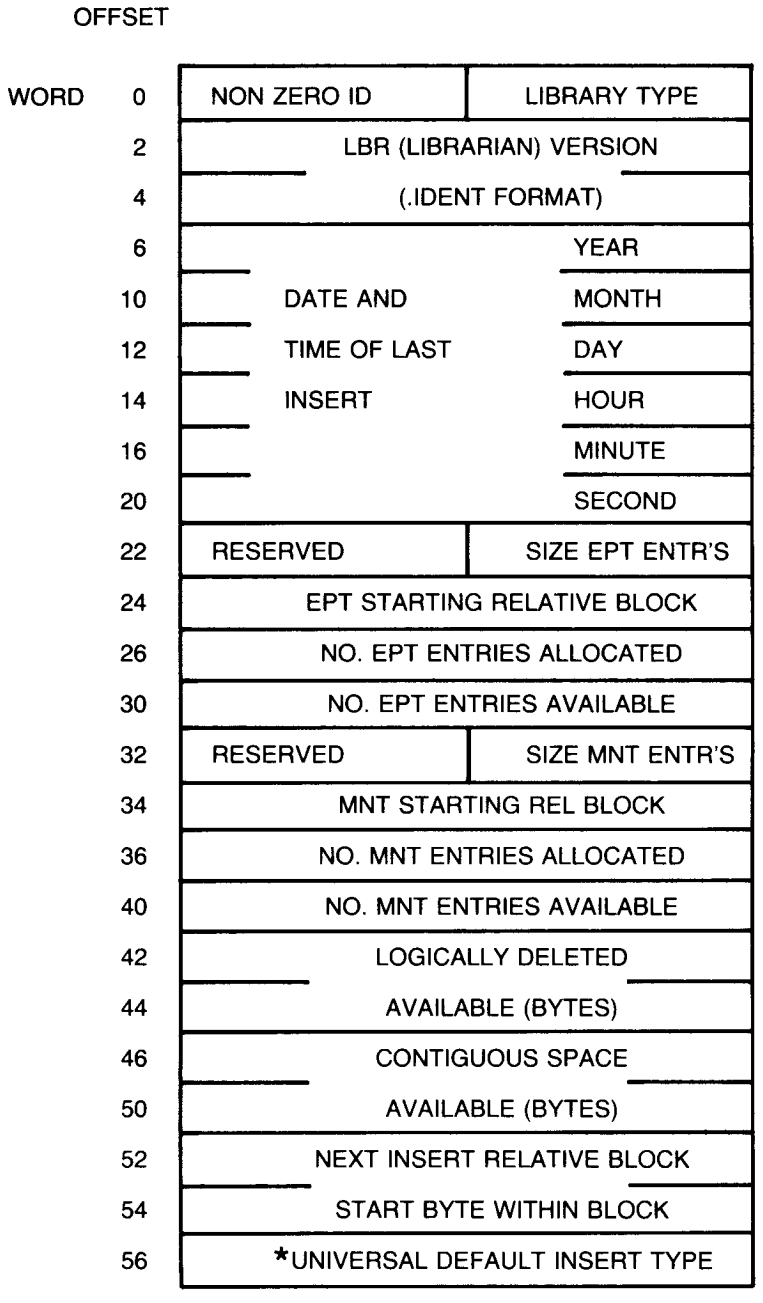
Figure B–2: Universal Library File Format



NOTE

All universal module headers and the first record of each universal module will start on a block boundary.

Figure B–3: Contents of Library Header



*Undefined for macro and object libraries

Figure B–4: Format of Entry Point Table Element

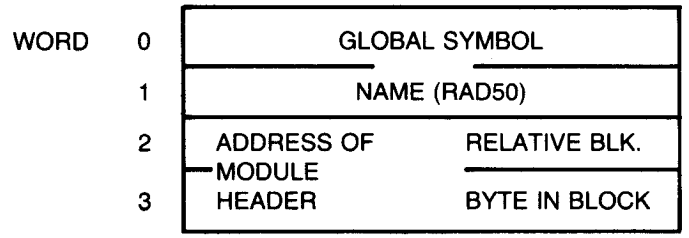


Figure B–5: Format of Module Name Table Element

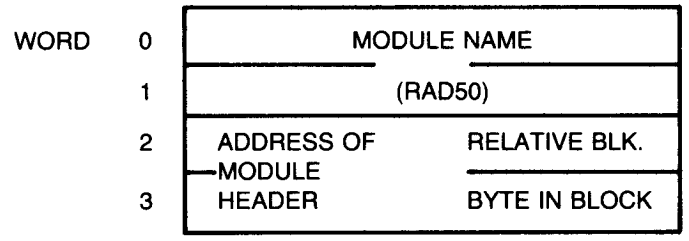


Figure B–6: Module Header Format

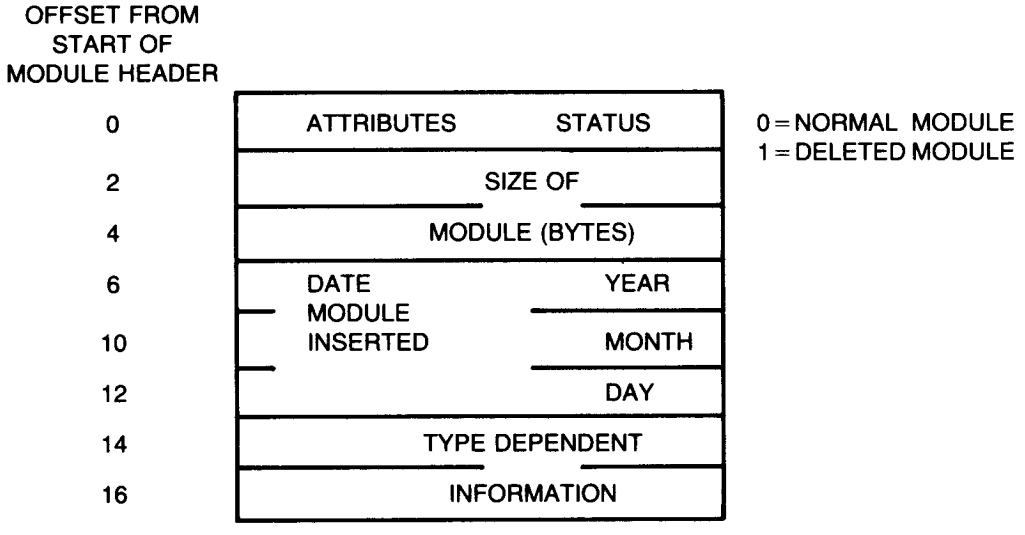
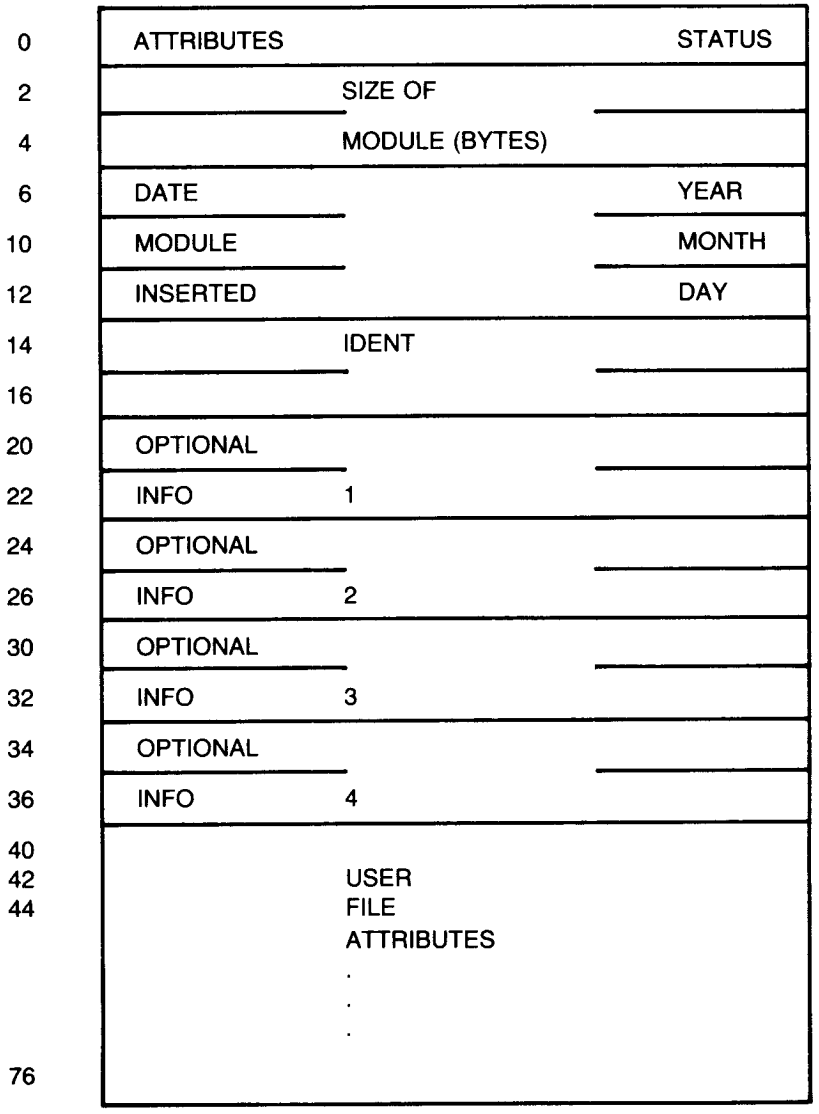


Figure B–7: Module Header Format for Universal Libraries



Index

A

ABS argument, 2–8t
Accessing utilities, 1–3
Adding a subroutine to modules, 4–7
AMA argument, 2–8t
Arguments
  for /EN and /DS switches, 2–8t
  for /LI and /NL switches, 2–7t
Assembly pass switch (/PA), 2–9 to 2–10

B

BEX argument, 2–7t
BIN argument, 2–7t
Build mode, with MAKSIL, 5–4

C

CCL, running MACRO–11 with, 2–4, 2–5
CCL command names, 1–4
  for RSX–based utilities, 1–4t
CDR argument, 2–8t
Checksum switch
  for file contents, 4–8
  numeric value of, 4–9
  in PAT command line, 4–4
CND argument, 2–7t
COM argument, 2–7t
Combining library functions, 3–23
Command
  CCL names for, 1–4
  indirect, for entering lines, 1–5, 1–6
  RUN, entering, 1–4
Command files
  indirect (LBR), 3–2
  indirect (MACRO), 2–5 to 2–6
Command line
  entering, 1–5
  fixing errors, 4–10
  format, 1–1
  LBR, 3–2
  PAT, specifying, 4–4
Command String format, 2–5
Compress switch, 3–2t, 3–3
Continuation lines in MACRO, 2–5
Correction file
  creating, 4–5
  used with PAT utility, 4–5
Create switch, 3–2t, 3–4
CSECT in PAT command line, 4–5
CTRL/C in MACRO DCL, 2–3
CTRL/Z
  with indirect command files, 2–6
  in MACRO DCL, 2–3

D

DCL
  command line expansion, 2–2
  command line format, 2–2
  /LIBRARY qualifier in MACRO, 2–3
  MACRO file specification, 2–2
  running MACRO in, 2–2 to 2–3
Default switch, 3–2t, 3–6
Defaults
  of file extensions, 1–3t
  of file specifications, 1–2t
Delete Global switch, 3–2t, 3–8
Delete switch, 3–2t, 3–5
Diagnostic error messages
  MACRO–11, A–1
  MAKSIL, 5–8
/DS switch in MACRO–11, 2–6t

E

Edit mode with MAKSIL, 5–4
/EN switch in MACRO–11, 2–6t
Entry Point switch, 3–2t, 3–8
Entry Point table, B–1
  format of elements, B–5f
  and library module referencing, B–1
Error codes for MACRO–11, A–1 to A–3
Error messages
  LBR, 3–24 to 3–32
  MACRO, 2–10 to 2–14
  MAKSIL, 5–5 to 5–8
  PAT, 4–9
Errors, types of
  command line, 4–10
  in file contents, 4–13
  in file format, 4–14
  file specification, 4–11, 4–12
  I/O, 4–12, 4–13
  internal software, 4–14
  storage allocation, 4–15
Extensions, file, default, 1–3t
Extract switch, 3–2t, 3–10
EXTSCT Task Builder option, 5–2t

F

Fatal errors
  effect on library files, 3–24
  MAKSIL error messages, 5–5 to 5–8
File contents
  determining, 4–8
  validating, 4–8, 4–9
File extensions, default, 1–3t
File specification
  defaults, 1–2t
  errors in, 4–11, 4–12
  example of, 1–2, 2–4
  MACRO–11 switches, 2–6
  MACRO I/O format, 2–4 to 2–5
  RSTS/E, 1–2
Files
  errors in contents, 4–13
  errors in format, 4–14
  indirect command (LBR), 3–2
  indirect command (MACRO), 2–5 to 2–6
  library, 3–1
  standard library format, B–2
  universal library format, B–3f
FPT argument, 2–8t
Function control switches, 2–8
Functions, combining library, 3–23

G

GBL argument, 2–8t

H

Header
  library, contents of, B–1, B–4f
  module, format of, B–2, B–5f

I

I/O, errors, 4–12 to 4–13
IDENT directive, to identify a module, B–2
Indirect command
  for entering lines, 1–5
  examples, 1–6
Indirect command files
  with LBR, 3–2
  with MACRO, 2–5 to 2–6
  nesting, 2–6
Information messages
  with MAKSIL, 5–9, B–4
  with PAT, 4–9, 4–10
Input/Output
  errors, 4–12
  MACRO–11 file specification format, 2–4
Insert switch
  in object and macro libraries, 3–2t, 3–11
Insert switch (Cont.)
  in universal libraries, 3–12
Internal software errors, 4–14

L

LBR, 3–1
  command line, 3–2
  error messages, 3–24 to 3–32
  files, sample, 3–16t, 3–17t
  files and formats, B–1
  restrictions, list of, 3–23
LBR switches, 3–2t
  Compress, 3–2t, 3–3
  Create, 3–2t, 3–4
  Default, 3–2t, 3–6
  Delete, 3–2t, 3–5
  Delete Global, 3–2t, 3–8
  Entry Point, 3–2t, 3–8
  Extract, 3–2t, 3–10
  Insert (object, macro, universal), 3–2t
  Insert (object and macro libraries), 3–11
  Insert (universal libraries), 3–12
  List, 3–2t, 3–13
  Modify Header, 3–2t, 3–14
  Replace, 3–2t, 3–16, 3–17
  Replace (object and macro libraries), 3–15
  Replace (universal libraries), 3–18, 3–19
  Selective Search, 3–2t, 3–20
  Spool, 3–2t, 3–20
  Squeeze, 3–2t, 3–21, 3–22
LC argument, 2–8t
LD argument, 2–7t
/LI switch in MACRO–11, 2–6t
Librarian Utility Program (LBR), 3–1
  CCL name for, 1–4t
  entering LBR command lines, 1–5
  files and formats, B–1
  to invoke, 1–4
Libraries, MACRO, 2–9
Library
  functions, combining, 3–23
  header, B–1
  resident, 5–3, 5–4
  standard, file format for, B–2f
  universal, file format for, B–3f
Library files, 3–1
  examples, 3–16t
  fatal error messages, 3–24
  sample output, 3–17t, 3–18t
  standard, format of, B–2f
  universal, format of, B–3f
Library header, B–1
  contents, B–4f
  and library status, B–1
/LIBRARY qualifier in MACRO DCL, 2–3

Library switch, MACRO, 2–9
List switches, 3–2t, 3–13
Listing control switches, 2–7
LOC argument, 2–7t
LSB argument, 2–8t

M

MACRO–11
  assembly process output files, 2–1
  CCL name for, 1–4t
  continuation lines in, 2–5
  DCL examples in, 2–2, 2–3
  default file types, 2–2
  defaults, 2–3
  diagnostic error messages, A–1
  error codes, A–1 to A–3
  error messages, 2–10 to 2–14
  file specification switches, 2–6
  I/O file specification format, 2–4 to 2–5
  with indirect command files, 2–5 to 2–6
  libraries, 2–9
  library switch, 2–9
  listing with Squeeze switch, 3–22f
  module names and LBR, 3–1
  running MACRO in DCL, 2–2
  switches, 2–6t
  utility program, 2–1
MAKSIL (Make Save Image Library)
  as CCL name, 1–4t
  creating a run-time system with, 5–1
  diagnostic messages, 5–8
  fatal error messages, 5–5 to 5–8
  information messages, B–4
  operating instructions, 5–3, 5–4
MC argument, 2–7t
.MCALL directive, 2–9
MD argument, 2–7t
ME argument, 2–7t
MEB argument, 2–7t
Messages
  command line errors, 4–10
  diagnostic errors, A–1
  fatal errors and library files, 3–24
  file content errors, 4–13
  file format errors, 4–13, 4–14
  file specification errors, 4–11
  I/O errors, 4–12, 4–13
  information, 4–9, 4–10
  internal software errors, 4–14
  LBR error, 3–24 to 3–32
  MACRO errors, 2–10 to 2–14
  MAKSIL diagnostic, 5–8
  MAKSIL fatal errors, 5–5 to 5–8
  MAKSIL informational, 5–9
  PAT errors, 4–9
Messages (Cont.)
  storage allocation error, 4–15
/ML switch, 2–9
  in MACRO–11, 2–6t
Modify Header switch, 3–2t, 3–14
Modules
  adding a subroutine to, 4–7
  header, described, B–2
  header, format, B–5f
  name table, B–1, B–2
  name table, format, B–5f
  names and LBR, 3–1
  object, updated with PAT, 4–6
  object, updated with Task Builder, 4–6
  overlaying lines in, 4–6
  updating with PAT, 4–2f, 4–3f

N

/NL switch in MACRO–11, 2–6t
/NOLIST in MACRO DCL, 2–3
/NOOBJ in MACRO DCL, 2–3
/NOSP switch in MACRO–11, 2–6t

O

Object module names and LBR, 3–1
Object modules
  patch utility, 4–1
  updated with PAT, 4–6
  updated with Task Builder, 4–6
Operating instructions for MAKSIL, 5–3, 5–4
Output library file, 3–17t, 3–18t
Overlaying lines in a module, 4–6

P

/PA switches, 2–9 to 2–10
/PA:1 switch in MACRO–11, 2–6t
/PA:2 switch in MACRO–11, 2–6t
PAR option
  Task Builder, 5–2t
  for various sized run-time systems, 5–2t
PAT command line, 4–4
  checksum switch, 4–4
PAT utility, 4–1
  correction file, 4–5
  information messages, 4–9, 4–10
  input file, 4–4
  kinds of error messages, 4–9
  starting, 1–4, 4–2
  for updating a module, 4–2f, 4–3f
  updating object modules, 4–6
  using PAT to apply updates, 4–4
Patch Object Module, CCL name for, 1–4t

Patches installing with PAT utility, 4–1
Physical address range, Task Builder option for, 5–2t
PNC argument, 2–8t
PSECT in PAT command line, 4–5

R

REG argument, 2–8t
Replace switch, 3–2t, 3–16, 3–17
  in object and macro libraries, 3–15
  in universal libraries, 3–18
Resident library, creating, 5–3, 5–4
RSX–based MACRO–11 assembler, 2–1
RSX–based utilities, CCL names for, 1–4t
RTS, 5–1. *See also Run-time system*
RUN command
  entering, 1–4
  MACRO–11, 2–4
Run-time system
  creating, 5–1
  PAR and STACK options, 5–2t

S

Selective Search switch, 3–2t, 3–20
SEQ argument, 2–7t
Software errors, internal, 4–14
/SP switch in MACRO–11, 2–6t
Spool switch, 3–2t, 3–20
Squeeze switch, 3–2t, 3–21, 3–22
  MACRO listing of, 3–22f
SRC argument, 2–7t
STACK option
  used with Task Builder, 5–2t
  for various sized run-time systems, 5–2t
Standard library file format, B–2f
STB, symbol table, 5–5
Storage allocation error, 4–15
Subroutines, added to a module, 4–7
Switches
  arguments for /EN and /DS, 2–8
  arguments for /LI and /NL, 2–7
  Checksum, 4–8, 4–9
  Compress, 3–2t, 3–3
  Create, 3–2t, 3–4
  Default, 3–2t, 3–6
  Delete, 3–2t, 3–5
  Delete Global, 3–2t, 3–8
  Entry Point, 3–2t, 3–8
  Extract, 3–2t, 3–10
  in file specification, 2–6
  function control, 2–8
  Insert (object, macro, universal), 3–2t
  Insert (object and macro libraries), 3–11
  Insert (universal libraries), 3–12
  LBR, 3–2t
Switches (Cont.)
  List, 3–2t, 3–13
  listing control, 2–7
  MACRO–11, 2–6t
  Modify Header, 3–2t, 3–14
  placement of in command string, 2–7
  Replace, 3–2t, 3–16, 3–17
  Replace (object and macro libraries), 3–15
  Replace (universal libraries), 3–18, 3–19
  Selective Search, 3–2t, 3–20
  Spool, 3–2t, 3–20
  Squeeze, 3–2t, 3–21, 3–22
SYM argument, 2–7t
Symbol table, appended to run-time system, 5–5
System MACRO library (LB:RSXMAC.SML), 2–9

T

Tables
  entry point, B–1
  entry point element format, B–5f
  module name, B–1
  symbol, appended to run-time system, 5–5
Task Builder
  options, for physical address range, 5–2t
  options, for virtual address range, 5–2t
  options, PAR and STACK, 5–2t
  processing files with, 4–7
  updating object modules with, 4–6
TOC argument, 2–7t
TTM argument, 2–7t

U

Universal libraries
  file format, B–3f
  sample files, 3–19t
Universal module names and LBR, 3–1
Updating modules with PAT, 4–1
Utilities
  accessing, 1–3
  command lines, entering, 1–4, 1–5
  command lines, format, 1–1
  LBR, files and formats, B–1
  Librarian, 1–4, 3–1, B–1
  MACRO–11, 2–1
  MAKSIL, 1–4, 5–1
  PAT, 1–4, 4–1
  RSX–based, CCL names for, 1–4

V

Virtual address range, Task Builder option for, 5–2t

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA and Puerto Rico call **800–258–1710**

In Canada call **800–267–6146**

In New Hampshire, Alaska or Hawaii call **603–884–6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary:
809–754–7575

RSTS/E
Programmer's Utilities Manual
AA–D749A–TC
AD–D749A–T1, T2

Reader's Comments

Note: This form is for document comments only. Digital will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement. _____

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number. _____

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Commercial Engineering Publications MK1-2/H3
DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK, N.H. 03054

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line