HITACHI MICROCOMPUTER SUPPORT SOFTWARE

H SERIES LIBRARIAN

USER'S MANUAL

HS6400LBCU1SE
ADE-702-087

Preface

This manual describes how to use the H Series Librarian. The manual is divided into the following eight sections.

Sections 1 and 2.....	Librarian functions
Section 3	Executing the Librarian
Section 4	Librarian options and subcommands
Section 5	Input to the Librarian
Section 6	Output from the Librarian
Section 7	Error messages
Section 8	Restrictions
Appendix A.....	Examples of Use of Librarian

Installation of the Librarian is covered in the Installation Guide supplied with the Librarian.

Refer to the following user's manuals for additional information on the other support tools in the H series cross system.

- H Series Linkage Editor User's Manual
- H8/300 Series Cross Assembler User's Manual
- H8/500 Series Cross Assembler User's Manual
- H32 Series Cross Assembler User's Manual
- SH Series Cross Assembler User's Manual
- H8/300 Series C Compiler User's Manual
- H8/500 Series C Compiler User's Manual
- H32 Series C Compiler User's Manual
- SH Series C Compiler User's Manual

Notes:

The following symbols have special meanings in this manual.

<item> : <specification item>

{ } : One of the items between the brackets is to be selected.

[ ] : The enclosed item is optional (i.e., can be omitted).

... : The preceding item can be replaced.

Δ : Blank space(s) or tab(s)

RET : Press the Return (Enter) key.

UNIX is an operating system administrated by the UNIX System Laboratories (United States).
MS-DOS is an operating system administrated by the Microsoft Corporation (United States).

Contents

Figures

Tables

Section 1. Overview

A program is usually developed by dividing it into functional modules and creating a separate source program for each module. Next, each source program module is compiled or assembled to create an object module. The object modules are then linked together using a linkage editor, resulting in an executable program.

The H Series Librarian introduced in this manual (hereafter called the Librarian) plays a vital role in this process. It brings together the many object modules output by the C compiler and assembler, as well as relocatable load modules output by the linkage editor, to make library files.

The Librarian provides the following advantages.

Simplified Module Management: The many modules making up a program (including relocatable load modules as well as object modules) are stored in a library file for the particular program. They can then be dealt with all at once. Moreover, it is possible to create generic library files that can be used later to streamline the creation of other programs.

A library file can be edited by adding, deleting, or replacing individual modules. In this way the modules can be kept up to date.

Enhanced Linkage: The Linkage Editor can search library files to find, extract, and link modules that define unresolved import symbols. Use of the library files thus makes linkage editing more efficient.

Section 2. Librarian Functions

2.1 Creating Library Files

This function makes it possible to create new library files, and to enter object modules output by the C compiler or assembler as well as relocatable load modules output by the linkage editor.
Figure 2-1 is an illustration of the library file creation concept.

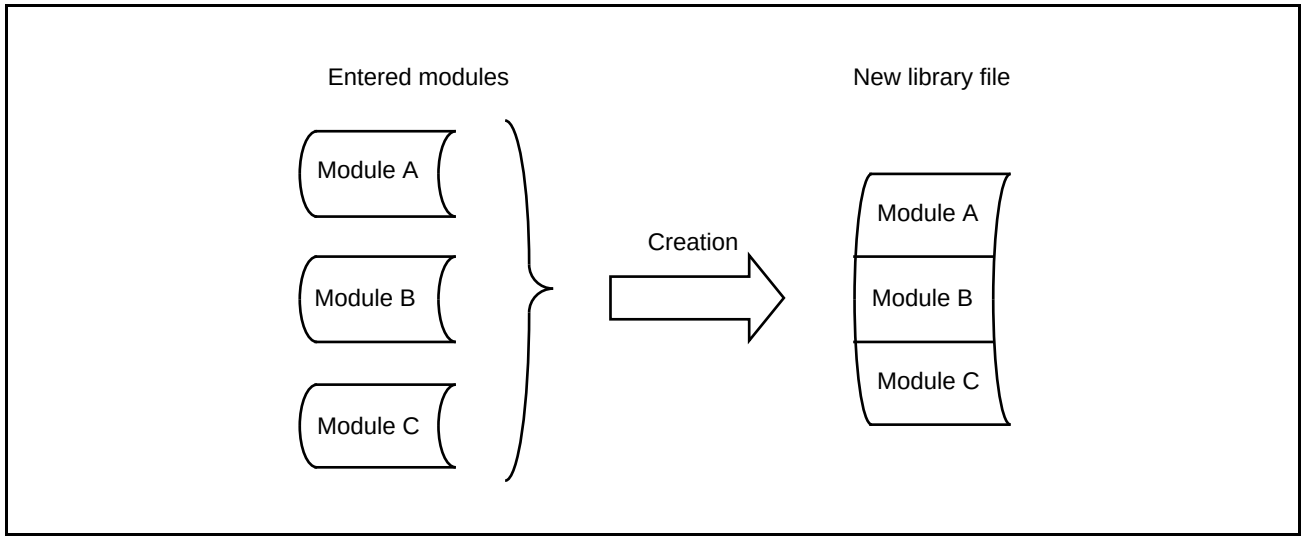


Figure 2-1. Creating a New Library File

2.2 Editing Existing Library Files

Modules can be added to, deleted from, or replaced in existing library files.

Adding Modules: Modules can be added to already existing library files. The concept of module addition is illustrated in Figure 2-2.

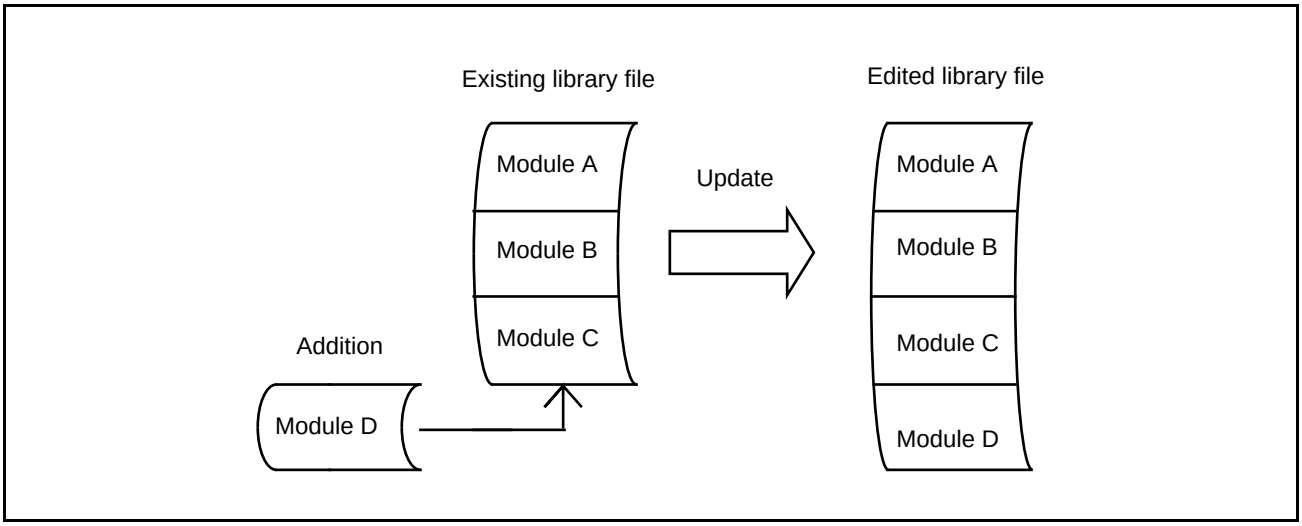


Figure 2-2. Adding a Module

Deleting Modules: Unnecessary modules can be deleted from existing library files. Figure 2-3 illustrates the module deletion concept.

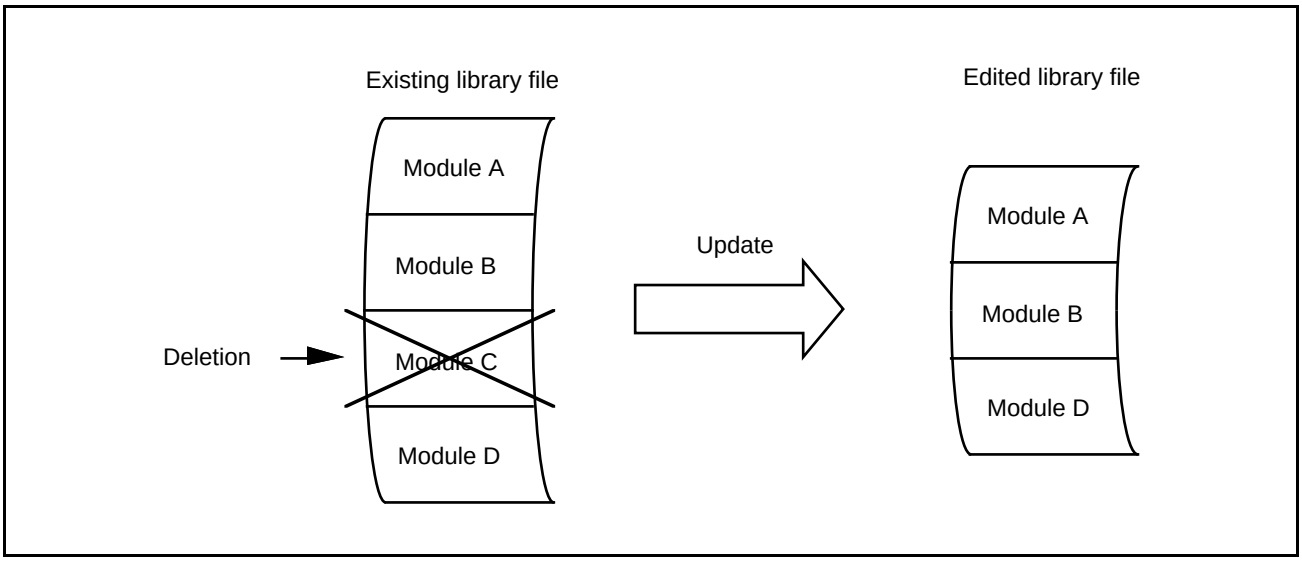


Figure 2-3. Deleting a Module

Replacing Modules: Modules in existing library files can be replaced with new modules. The concept of module replacement is illustrated in Figure 2-4.

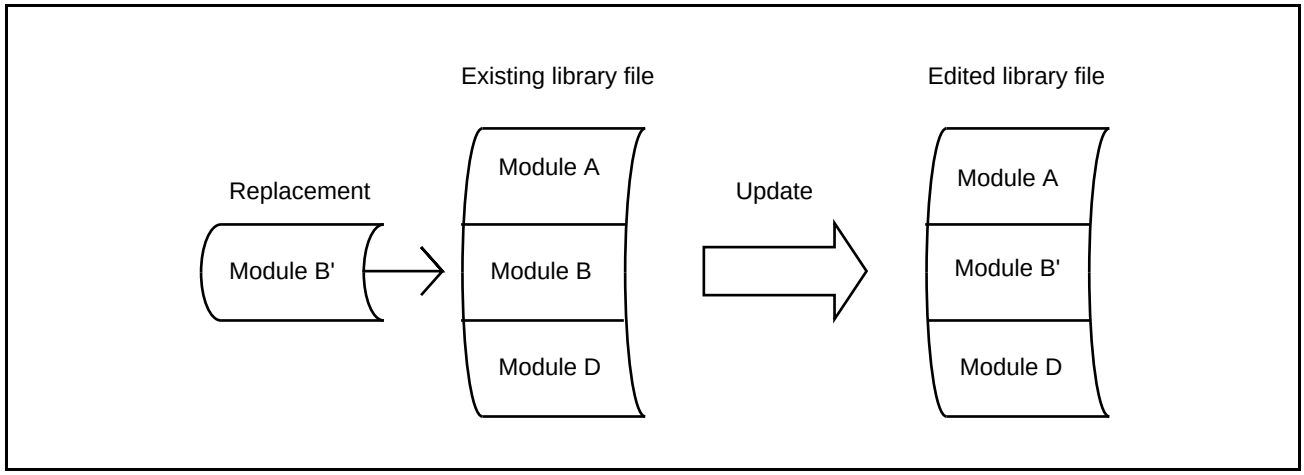


Figure 2-4. Replacing a Module

2.3 Extracting Modules from a Library File

Modules can be extracted from existing library files and used to create new library files. The concept of module extraction is illustrated in Figure 2-5.

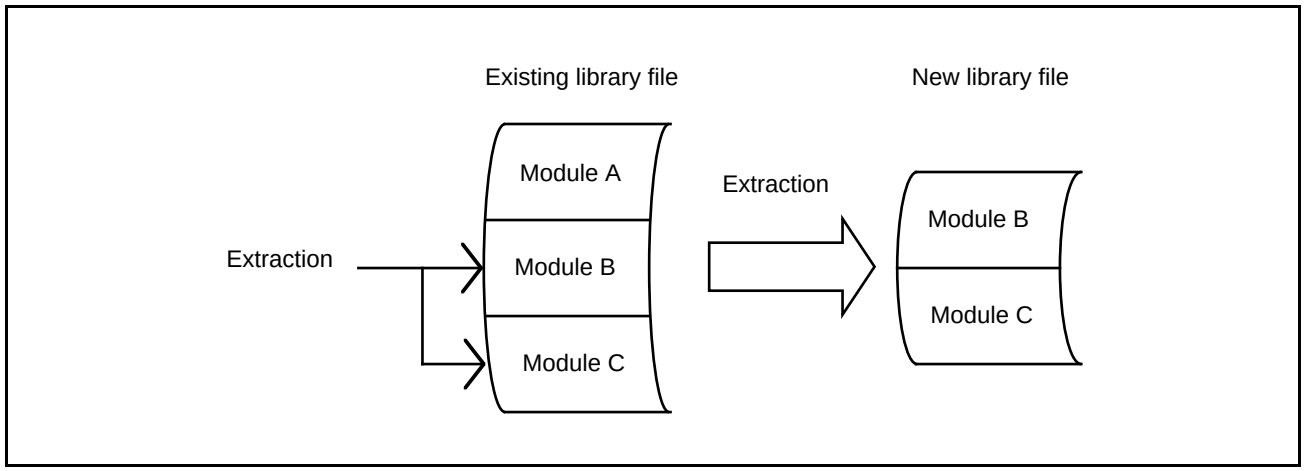


Figure 2-5. Extracting Modules

2.4 Displaying the Contents of a Library File

A librarian list giving information about the modules and export symbols in a library file can be output to a standard output device or a list file. A librarian list tells when the library file was created and when it was last revised, indicates when each module was stored, and gives the names of export symbols and other useful information.

For further details, see section 6.2, Librarian Lists.

Section 3. Executing the Librarian

To execute the Librarian, first start the Librarian by entering a command line. The command line specifies the name of the library file to be edited and various options, which give instructions to the Librarian. If these instructions are sufficient, the Librarian can be executed using the command line alone. If further instructions are needed, they can be given in subcommands.

Command Line Execution: The Librarian can be executed simply by specifying a library file and options on the command line. The method is useful when library editing is relatively straightforward.

Subcommand Execution: The Librarian can also be executed by entering both a command line and subcommands. The subcommands specify input and output files and parameters that control the Librarian. This method is useful for specifying a large number of files or modules, or for editing two or more library files together. Subcommands can be entered interactively, or from a subcommand file. Details are given in section 3.3, Executing by Subcommands.

3.1 Command Line Format

The following format is used for the Librarian command line.

UNIX System:
lbr[Δ[<library file name>][[Δ]-<option name>[[Δ]-<option name>...]]] RET

MS-DOS system:
lbr[Δ[<library file name>][[Δ]/<option name>[[Δ]/<option name>...]]] RET

Command Name: "lbr" is the command that starts the Librarian.

Library File Name: To edit or extract modules from an existing library file, type the name of the library file in the command line.

Option Names: Each option name must start with a hyphen (-) at UNIX system or with a slash (/) at MS-DOS system. One or more spaces or tabs may also be used to separate an option name from a preceding option name or library file name, but these spaces or tabs are not required. Option names are described in detail in section 4, Librarian Options and Subcommands. The Librarian edits the library file according to the order in which the options are specified.

Specifying the Mode of Execution: The content of the command line determines whether the Librarian will be executed by the command line specifications only, or by subcommands. See Table 3-1.

The library file name and each option name must be specified in 128 characters or less (not including the final carriage return).

Table 3-1. How Command Line Specification Determines the Form of Execution

	Option Specification			
Library File Name Specification	**No Option Specified**	**SUBCOMMAND Option Specified**	**CREATE Option Specified**	**Option Other than CREATE or SUBCOMMAND Specified**
Library file name specified	—	—	—	Executed by specifying command line
No library file name specified	Executed by specifying subcommands	Executed by specifying subcommands	Executed by specifying command line	—

Notes:
1. For SUBCOMMAND and CREATE options, see section 4, Librarian Options and Subcommands.
2. The combinations of option and library file names indicated by dashes (—) are not permitted. An error will occur, and the librarian will not be executed.

3.2 Executing by Command Line

With this method, the Librarian is executed according to the information specified in the command line alone. Editing procedures and other conditions are specified to the Librarian in the form of options. When the editing process is straightforward and simple, command line specification is sufficient for creating or updating a library. Examples of execution by command line are given below.

EXAMPLE 1 (UNIX system):

```
% lbrΔ-CREATE=syslib.lib-ADD=obj00.obj,prg.lib 3
```

1 **2**

1 Creates a new library file named syslib.lib.
2 Adds the modules in object module file obj00.obj and library file prg.lib to syslib.lib.

The CREATE option by itself will not create a library file unless modules are added using the ADD option.

EXAMPLE 2 (MS-DOS system):

```
% lbrΔsyslib.lib/ADD=obj00.obj/DELETE=mod13
```

1 **2** **3**

1 Designates library file syslib.lib as the file to be edited.
2 Adds the module in object module file obj00.obj to syslib.lib.
3 Deletes existing module mod1 from syslib.lib.

3.3 Executing by Subcommands

Since the number of characters that can be typed on the command line is limited, the command line may not be able to accommodate a large number of specifications. In such cases, subcommands are used to execute the Librarian. Subcommands can be input interactively, one at a time, from the keyboard or other standard input device. Alternatively, a subcommand file consisting of a group of subcommands can be created in advance, and subcommands can be input from this subcommand file.

3.3.1 Executing in Interactive Mode

When no library file is specified in the command line and there are no option specifications, execution proceeds in interactive mode. A colon (:) appears on the screen as a prompt, indicating that the Librarian is waiting for a subcommand to be input. In this way you can enter the necessary subcommands. This method is useful when the number of subcommands is relatively small, or when you want to check Librarian lists as you enter the subcommands.

An example of execution by interactive input of subcommands is given below. Functions of the subcommands listed here are detailed in section 4, Librarian Options and Subcommands.

EXAMPLE:
% lbr RET...**1**

: CREATEΔprg.lib RET...**2**

: ADDΔmain.obj RET...**3**

: ADDΔsend.obj,receive.obj,exchange.obj RET...**4**

: ADDΔaccount.obj RET...**5**

: LISTΔ(S) RET...**6**

: EXIT RET...**7**

1 Starts the Librarian in interactive mode.
2 Creates a new library file named prg.lib
3 Adds the module in main.obj to prg.lib.
4 Adds the modules in send.obj, receive.obj and exchange.obj to prg.lib.
5 Adds the module in account.obj to prg.lib.
6 Outputs a librarian list, including symbol information, to the standard output device.
7 Terminates the Librarian operation.

3.3.2 Executing from a Subcommand File

This method uses a subcommand file that was created in advance and that contains the subcommands necessary for Librarian operations. This subcommand file is then specified on the command line as a parameter of the SUBCOMMAND option. This method is useful when many subcommands must be specified, or when the same editing process is carried out repeatedly. It eliminates the need to input subcommands from the keyboard or other standard input device each time.

Use an editor to create the subcommand file. An example of execution from a subcommand file is given below. Functions of the subcommands listed here are detailed in section 4, Librarian Options and Subcommands.

% lbrΔ-SUBCOMMAND=prglib.sub RET... **1**

Contents of subcommand file prglib.sub:	
CREATEΔfunction.lib	... **2**
ADDΔsin.obj,cos.obj,tan.obj	... **3**
ADDΔasin.obj,acos.obj,atan.obj	... **4**
ADDΔhsin.obj,hcos.obj,htan.obj	... **5**
ADDΔlog.obj,log10.obj	... **6**
EXIT	... **7**

1 Starts the Librarian and inputs subcommands from subcommand file prglib.sub.
2 Creates a new library file function.lib.
3 Adds the modules in object module files sin.obj, cos.obj and tan.obj to function.lib.
4 Adds the modules in object module files asin.obj, acos.obj and atan.obj to function.lib.
5 Adds the modules in object module files hsin.obj, hcos.obj and htan.obj to function.lib.
6 Adds the modules in object module files log.obj and log10.obj to function.lib.
7 Terminates Librarian operations.

3.4 Terminating Librarian Operations

When the Librarian terminates operations, it gives the system a return code indicating an error level. The return code can be used to control the execution of a command file. The error levels and their return codes are:

UNIX system:

Normal completion	0
Warning	0
Error	1
Fatal error	1

MS-DOS system:

Normal completion	0
Warning	0
Error	2
Fatal error	4

Section 4. Librarian Options and Subcommands

Options and subcommands tell the Librarian what editing operations to perform. The three main functions of options and subcommands are file control, execution control, and list display. These functions can be used individually or in combination to create and edit library files.

Options and subcommands have the same names and equivalent functions, but are specified in different formats. Moreover, there are some specifications which can be made only with options, and others only with subcommands. Sections 4.1, Option and Subcommand Formats, and 4.2, List of Options and Subcommands, must accordingly be read carefully. Option and subcommand functions are outlined below.

File Control Functions: File control functions indicate the name of the library file to be edited, or the name of a library file to which extracted modules are to be output.

Execution Control Functions: Execution control functions instruct the Librarian to perform editing operations, or terminate its processing. These functions are used, for example, to input subcommands from a subcommand file, to create a new library file, or to update a library file.

List Display Functions: List display functions are used to display information such as names of modules stored in a library file, or export symbol names.

Note: The examples are written for UNIX system, please write slash (/) instead of hyphen (–) for MS-DOS system and write all options and commands in capital letters.

4.1 Option and Subcommand Formats

Each option or subcommand consists of a name and parameters, which together must not exceed 128 characters.

Option and Subcommand Structure: Options and subcommands differ as to the way of separating the name from the parameters. Options use an equals sign (=), while subcommands use one or more spaces or tabs.

Option format

<Name>=<parameters>

Subcommand format

<Name>Δ<parameters>

EXAMPLES:

–OUTPUT=lbf ..... option
OUTPUTΔlbfsubcommand

In these examples, OUTPUT is the name, and lbf is the parameter.

(a) Name

The name gives the name of the option or subcommand.

(b) Parameters

The parameters give the names of files,*1 module,*2 etc. on which the option or subcommand operates. There are different requirements and methods of specification depending on the type of option or subcommand. For details, refer to section 4.3, File Control, section 4.4, Execution Control, and section 4.5, List Display.

Notes:

1. A file name consists of three parts: the path name, main file name, and file type.

   If the file type is omitted, a file type is assumed as follows.

   Library file ..... .lib
   Object module file ..... .obj
   Relocatable load module file ..... .obj
   Subcommand file ..... .sub
   List file ..... .lst

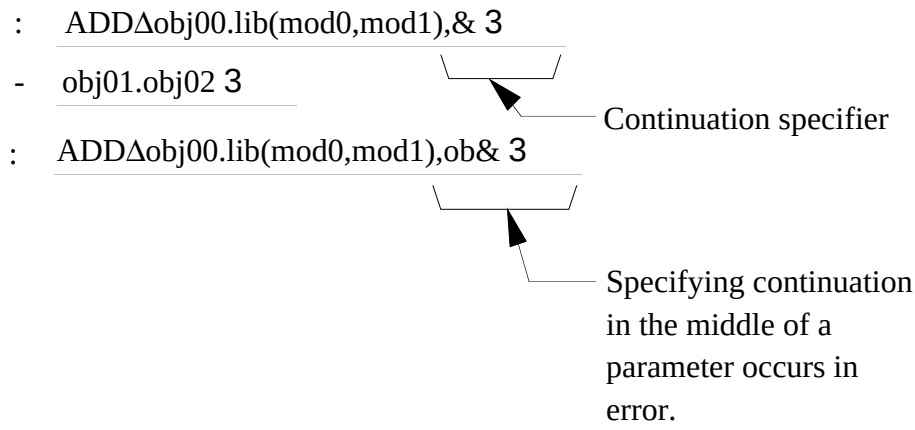
2. A module name is the name defined in an object module or relocatable load module. In module names, capital letters are distinguished from small letters. The pairs of names below, for example, are treated as different names.

EXAMPLES: modul1 ◄——► MODUL1

Continuation Specification in a Subcommand: When a subcommand is too long to be specified on one line, a continuation specifier is used. This consists of an ampersand (&) at the end of the line. It must always be placed between two parameters; if it is placed within a parameter, it will not be treated as a continuation specifier. Also, if a character (including a space or tab) is typed after the ampersand, an error will occur and the subcommand will not be continued.

In interactive input of subcommands, a hyphen (-) appears as a prompt for further input after continuation has been specified.

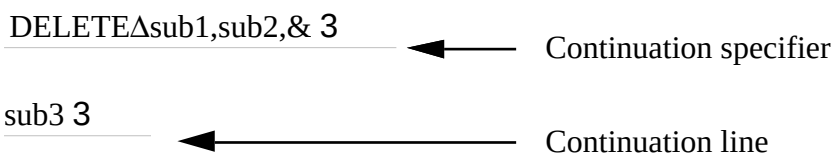
EXAMPLES:



A subcommand line in a subcommand file can be continued in the same way. The line after the line with the continuation specifier becomes the continuation line.

EXAMPLE:

Subcommand file



Specifying Comments in a Subcommand File: A comment specifier is used to place notes or other comments in a subcommand file. The specifier is a semicolon (;) placed on a subcommand line, indicating that the rest of the line is a comment. If the semicolon follows a subcommand name or parameter, it must be separated by at least one space or tab.

If the semicolon is placed at the beginning of a subcommand line, the entire line is treated as a comment.

EXAMPLES:

```
; EXAMPLE OF LIBRARIAN SUBCOMMAND
... the entire line is a comment.
LIBRARYΔsyslibΔ; INDICATES LIBRARY FILE
... INDICATES LIBRARY FILE is a comment.
ADDΔmodule.obj;abc
... module.obj;abc is treated as a single parameter
    abc is not treated as a comment.
```

4.2 List of Options and Subcommands

There are eight options and eleven subcommands, as listed in Table 4-1.

Table 4-1. List of Options and Subcommands

No.	Type	Name	Function	Opt.	Sub.	Section
1	File control	LIBRARY	Specifies the library file to be edited	No	Yes	4.3.1
		OUTPUT	Specifies an output library file	Yes	Yes	4.3.2
2	Execution control	SUBCOMMAND	Specifies a subcommand file	Yes	No	4.4.1
		CREATE	Creates a library file	Yes	Yes	4.4.2
		ADD	Adds modules	Yes	Yes	4.4.3
		REPLACE	Replaces modules	Yes	Yes	4.4.4
		DELETE	Deletes modules	Yes	Yes	4.4.5
		EXTRACT	Extracts modules	Yes	Yes	4.4.6
		END	End of subcommand input	No	Yes	4.4.7
		EXIT	End of Librarian operations	No	Yes	4.4.8
		ABORT	Aborts Librarian operations	No	Yes	4.4.9
3	List display	LIST	Displays contents of library file	Yes	Yes	4.5.1

Notes: 1. The underlined letters of a name are the shortest permissible abbreviated form.
2. The Opt. and Sub. columns indicate whether a name is available as an option or subcommand.

Abbreviating Option and Subcommand Names: Names of options and subcommands may be abbreviated to the point where the name can still be distinguished from other names. As an example, consider the name EXTRACT.

E ... Cannot be distinguished from EXIT or END, so an error occurs.
EX ... Cannot be distinguished from EXIT, so an error occurs.
EXT ... Recognized as EXTRACT.
EXTRA ... Recognized as EXTRACT.
EXTRACT ... Recognized as EXTRACT.
EXTRACTS ... No such name, so an error occurs.

Interrelation among Different Options and Subcommands: Once an option or a subcommand has been specified, other options or subcommands with conflicting functions cannot be specified. This interrelationship is shown in Table 4-2.

Table 4-2. Interrelation among Options and Subcommands

	Later Specification											
Specified Option/ Subcommand	**SUBCOMMAND**	**LIBRARY**	**CREATE**	**ADD**	**REPLACE**	**DELETE**	**EXTRACT**	**OUTPUT**	**LIST**	**END**	**EXIT**	**ABORT**
SUBCOMMAND	×	×	×	×	×	×	×	×	×	×	×	×
LIBRARY	×	×	×	o	o	o	o	o	o	o	o	o
CREATE	×	×	×	o	o	o	×	×	o	o	o	o
ADD	×	×	×	o	o	o	×	×	o	o	o	o
REPLACE	×	×	×	o	o	o	×	×	o	o	o	o
DELETE	×	×	×	o	o	o	×	×	o	o	o	o
EXTRACT	×	×	×	×	×	×	o	o	o	o	o	o
OUTPUT	×	×	×	×	×	×	o	×	o	o	o	o
LIST	×	×	×	o	o	o	o	o	o	o	o	o
END	×	o	o	×	×	×	×	×	×	×	o	o
EXIT	×	×	×	×	×	×	×	×	×	×	×	×
ABORT	×	×	×	×	×	×	×	×	×	×	×	×

o : Later specification enabled.

× : Later specification disabled, since it conflicts with already specified option or subcommand.

EXAMPLES:

```
% lbrΔ-SUBCOMMAND=funlib.sub-LIST ↵
```

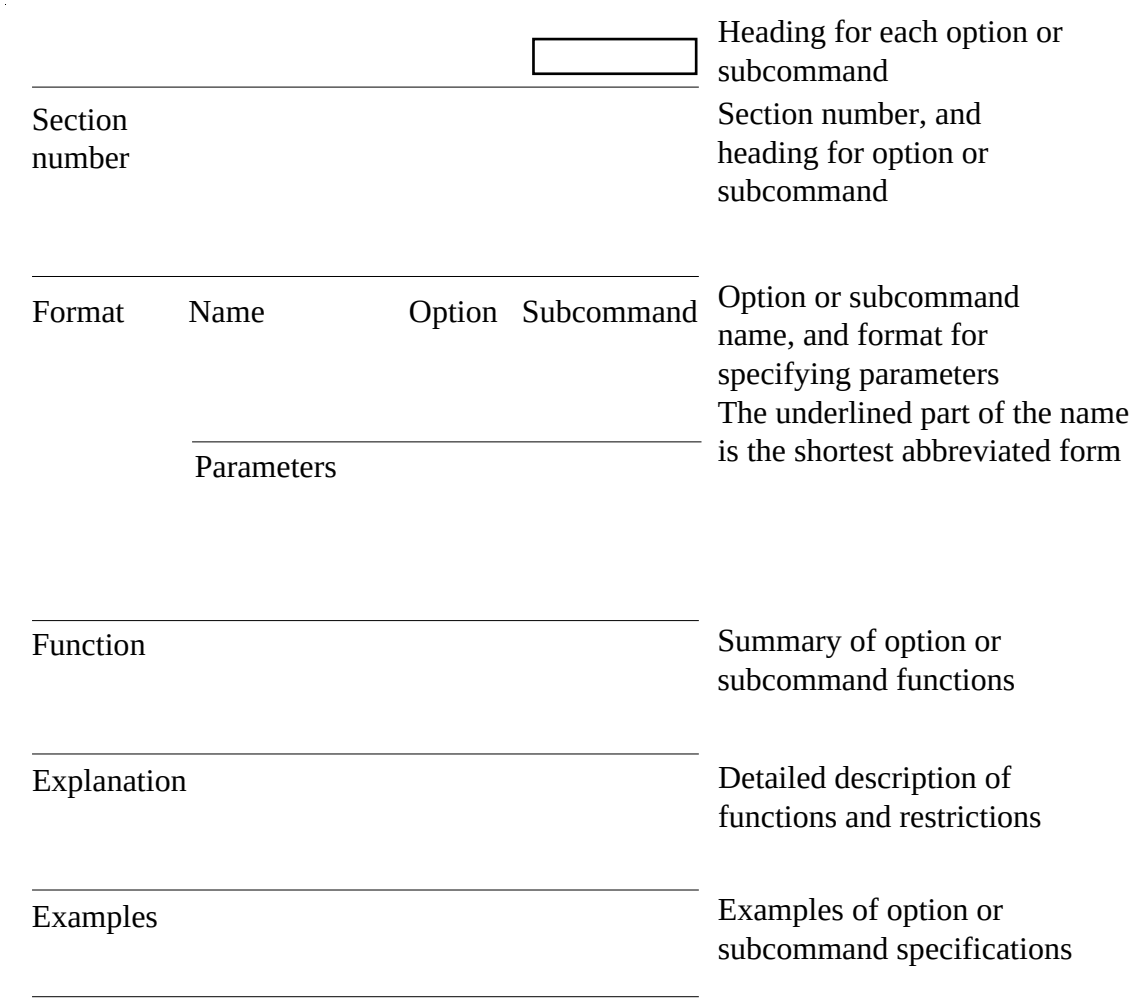
This results in an error since no other option can be specified after a SUBCOMMAND option.

```
% lbr ↵
: LIBRARYΔfunclib.lib ↵
  CREATEΔnewlib.lib ↵
: END ↵
: LIST ↵
: EXIT ↵
```

A CREATE subcommand cannot be specified after a LIBRARY subcommand. An error occurs, and the CREATE subcommand is ignored.

Specifying a LIST subcommand after an End subcommand, occurs in an error. After END, only the LIBRARY, CREATE, EXIT or ABORT subcommand is valid.

In the following sections, the format below is used to describe each option and subcommand.



4.3 File Control

4.3.1 LIBRARY — Specifies the library file to be edited.

Format	Name	LIBRARY	Option	Subcommand
			No	Yes
	Parameters	<Library file name>		

Function		Specifies an existing library file for editing.
Explanation	(1)	This subcommand is specified at the beginning of an editing operation that edits an existing library file or extracts modules from an existing library file.
	(2)	Only a library file created by this Librarian can be specified
	(3)	When no file type is specified as part of the library file name, the type is assumed to be .lib.
	(4)	This subcommand cannot be used together with the CREATE subcommand, which specifies creation of a new library file.
	(5)	If, as the result of editing an existing library file, the number of modules becomes zero, the library file will not be updated.
	(6)	The access right to the updated library file is the same as the access right to a newly created file. Note that the access right prior to the update is not preserved.
Examples	LIBRARY∆syslib Specifies editing of the library file syslib.lib.	

4.3.2 OUTPUT — Specifies an output library file.

Format	Name	OUTPU	Option	Subcommand
			Yes	Yes
	Parameters		Option	<Library file name>
			Subcommand	<Library file name> [(S) / (U)]

Function

Specifies a library file for output of extracted modules.

Explanation

(1) Specify the OUTPUT option or subcommand whenever a module is to be extracted from an existing library file.

(2) Specify a new library file name. When no file type is specified as part of the library file name, the type is assumed to be .lib.

(3) The attribute (S) or (U) is assigned to the output file. If unspecified, the attribute is assumed to be (U).

(S) ... System library

(U) ... User library

This attribute determines the order of priority in which library files are searched by the Linkage Editor. A user library has higher search priority. The (S) and (U) parameters cannot be included when OUTPUT is specified as an option.

(4) OUTPUT may be specified either before or after the EXTRACT option or subcommand, which specifies extraction of modules.

(5) OUTPUT cannot be used together with the CREATE, ADD, DELETE, or REPLACE options or subcommands.

(6) When the number of extracted modules is zero, the library file specified by the OUTPUT option or sub-command is not created.

Examples

–OUTPUT=PROG86

Modules extracted using the EXTRACT subcommand will be output as a user library to a file named prog86.lib.

OUTPUTΔclib.o(S)

Modules extracted using the EXTRACT subcommand will be output as a system library to a file named clib.o.

SUBCOMMAND

4.4 Execution Control

4.4.1 SUBCOMMAND — Specifies a subcommand file.

Format	Name	SUBCOMMAND	Option	Subcommand
			Yes	Yes
	Parameters	<Subcommand file name>		

Function		Inputs subcommands from a specified file.
Explanation	(1)	Inputs and processes subcommands from a specified subcommand file one at a time.
	(2)	When an EXIT subcommand or the end of the subcommand file (EOF) is detected, Librarian operations end.
	(3)	When no file type is specified as part of the file name, the type is assumed to be .sub.
	(4)	A SUBCOMMAND option cannot be specified more than once, or used together with other options.
Examples	–SUBCOMMAND=makelib Subcommands are input from the subcommand file makelib.sub for use in editing a library file.	

4.4.2 CREATE — Creates a library file.

Format	Name	CREATE	Option	Subcommand
			Yes	Yes
	Parameters		Option	<Library file name>
			Subcommand	<Library file name> {(S) / (U)}

Function Creates a new library file.

Explanation

(1) Specified at the beginning of a group of options or subcommands ending with END or EXIT.

(2) Specify a new library file name. When no file type is specified as part of the library file name, the type is assumed to be .lib.

(3) The attribute (S) or (U) is assigned to the output file. If unspecified, the attribute is assumed to be (U).

(S) ... System library

(U) ... User library

This attribute determines the order of priority in which library files are searched by the Linkage Editor. A user library has higher search priority. The (S) and (U) parameters cannot be included when CREATE is specified as an option.

(4) CREATE cannot be used together with the LIBRARY subcommand.

(5) If the number of modules is zero, no library file is created.

Examples

–CREATE=userlib.lib

Creates userlib.lib as a new user library.

CREATE∆sislib(S)

Creates sislib.lib as a new system library.

CREATE∆datax

Creates datax.lib as a new user library.

4.4.3 ADD — Adds modules.

Format	Name	ADD	Option	Subcommand
			Yes	Yes
	Parameters	Option	{<Object module file name> / <Relocatable load module file name> / <Library file name>} [,...]	
		Subcommand	{<Object module file name> / <Relocatable load module file name> / <Library file name>[(<module name>[,...])]} [,...]	

Function Adds modules from specified files to a library file.

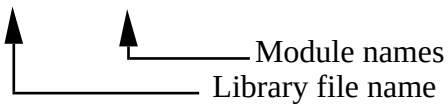
Explanation

(1) ADD is used to store modules in a new library file, or add modules to an existing library file.

(2) When only a file name is specified, if no file type is specified, the type is assumed to be .obj. When a module name is specified after a file name, the file is assumed to be a library file, so if no file type is specified, the type is assumed to be .lib.

(3) When only certain modules from a library file are to be added, specify the module names after the library file name. Up to 10 module names may be specified. However module names can not be included when ADD is specified as an option.

EXAMPLE: ADD lbf (m1,m2,m3)



(4) When modules in a library file are specified, the specified module names are sorted in alphabetical order and the modules are added in that order. They are not added in the order in which specified.

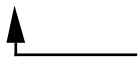
EXAMPLE: ADD lbf (e, a, d, c, b)
5, 1, 4, 3, 2... Order in which modules are added

(Continued on next page)

Explanation (cont)

(5) When the names of modules in a library file are not specified, all modules in the library file are added.

EXAMPLE: ADD lbf.lib



(6) When a module to be added has the same name as a module already in the library file being edited, or when an externally defined symbol defined in the module to be added has the same name as an externally defined symbol in the library file being edited, a warning message is displayed and the module is not added.

(7) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand is a convenient way of confirming which modules are stored in a library file.

(8) ADD cannot be used together with EXTRACT or OUTPUT options or subcommands.

(9) Errors will occur and the parameters after the error occurs will not be processed when:

(a) A specified file does not exist.
(b) A specified module does not exist in a library file.
(c) The content of the specified file is invalid.
(d) The number of modules to be stored exceeds 32,767.
(e) Memory capacity is insufficient to add more modules.
(f) The number of input files exceeds 12.

Examples

–ADD=mod1,mod2,modx.o

Adds all modules from the object module files mod1.obj, mod2.obj and modx.o.

ADDΔiofnc(keyin,crtout)

Adds the two modules keyin and crtout from the library file iofnc.lib.

ADDΔsyslib.lib

Adds all modules from the library file syslib.lib.

4.4.4 REPLACE — Replaces modules.

Format	Name	REPLACE	Option	Subcommand
			Yes	Yes
	Parameters	Option	{ <Object module file name> / <Relocatable load module file name> / <Library file name> } [,...]	
		Subcommand	{ <Object module file name> / <Relocatable load module file name> / <Library file name>[(<module name>[,...])] } [,...]	
Function		Substitutes modules in a specified file for modules of the same name in the library file being edited.		

Explanation

(1) When a module in the library file being edited has the same name as a module in the specified file, the former is replaced by the latter. If there is no module with the same name in the library file being edited, the module is simply added as with the ADD option or subcommand.

(2) When only a file name is specified, if no file type is specified, the type is assumed to be .obj. When a module name is specified after a file name, the file is assumed to be a library file, so if no file type is specified, the type is assumed to be .lib.

(3) To substitute only certain modules from a library file, specify the module names after the library file name. Up to 10 module names may be specified. However, module names cannot be included when REPLACE is specified as an option.

EXAMPLE: REPLACE lbf (m1,m2,m3)

Module names
Library file name

(4) When modules in library files are specified, the specified module names are sorted in alphabetical order and modules are replaced in that order. They are not replaced in the order in which specified.

EXAMPLE: REPLACE lbf (e, a, d, c, b)

5, 1, 4, 3, 2 ...Order of replacement

(Continued on next page)

Explanation

(cont)

(5) When the names of modules in a library file are not specified, all modules in the file are substituted.

EXAMPLE: REPLACE lbf.lib

(6) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand is a convenient way of confirming which modules are stored in a library file.

(7) REPLACE cannot be used together with EXTRACT or OUTPUT options or subcommands.

(8) The following cases will result in error, and the parameters after the error position will not be processed.

(a) A specified file does not exist.

(b) A specified module does not exist in a library file.

(c) The content of the specified file is invalid.

(d) The number of modules to be stored exceeds 32,767.

(e) Memory capacity is insufficient for the substitution to be performed.

(f) The number of input files exceeds 12.

(9) The process of replacing a module involves deleting the module of the same name in the library file being edited, then inputting the module from the file specified by the REPLACE option or subcommand and storing it in the library file. The following special caution is thus required: If a module to be substituted contains an externally defined symbol already defined in another module in the library file, the old module will be deleted, but the replacement module will not be stored.

(Continued on next page)

Examples –REPLACE=userlib.lib

All modules in the library file userlib.lib are stored in the library file being edited, replacing modules with the same name.

REPLACE∆loadx.rel,loady.rel

The two modules in the relocatable load module files loadx.rel and loady.rel are substituted for modules of the same name in the library file being edited.

REPLACE∆datax(member),omf

The module named member in library file datax.lib, and the module in the object module file omf.obj, are substituted for modules of the same name in the library file being edited.

4.4.5 DELETE — Deletes modules.

Format	Name	DELETE	Option	Subcommand
			Yes	Yes
	Parameters	<Module name> [,...]		

Function

Deletes specified modules from the library file being edited.

Explanation

(1) If a specified module does not exist in the library file, an error occurs, and the parameters after the error occurrence are not processed.

(2) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand is a convenient way of confirming which modules are stored in a library file.

(3) DELETE cannot be used together with EXTRACT or OUTPUT options or subcommands.

Examples

–DELETE=inchar,outchar

Deletes the two modules inchar and outchar.

DELETEΔdatatbl,sort

Deletes the two modules datatbl and sort.

4.4.6 EXTRACT — Extracts modules.

Format	Name	EXTRACT	Option	Subcommand
			Yes	Yes
	Parameters	<Module name> [,...]		

Function

Extracts specified modules from the library file being edited.

Explanation

(1) The extracted modules are output in library file format with the file name specified by the OUTPUT option or subcommand.

(2) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand is a convenient way of confirming which modules are stored in a library file.

(3) If a specified module does not exist in the library file, an error occurs, and the parameters after the error occurrence are not processed.

(4) EXTRACT cannot be used together with the CREATE, ADD, DELETE or REPLACE options or subcommands.

Examples

–EXTRACT=add,sub,mul,div

Extracts the four modules add, sub, mul, and div from the library file being edited.

EXTRACTΔalpha,upper,lower,digit,cntrl

Extracts the five modules alpha, upper, lower, digit, and cntrl from the library file being edited.

4.4.7 END — Specifies end of subcommand input.

Format	Name	END	Option	Subcommand
			No	Yes
	Parameters	None		

Function		Outputs a newly created or updated library file.
Explanation	(1)	When more than one library file is edited in one Librarian execution, the editing of each library file is terminated by an END subcommand.
	(2)	Specification of the END subcommand causes the Librarian to output the edited library file. If, however, the number of modules stored in the library file is zero, the library file is not created or updated.

Examples

```
END
```

Outputs a library file.

EXIT

4.4.8 EXIT — Specifies end of Librarian operations.

Format	Name	EXIT	Option	Subcommand
			No	Yes
	Parameters	None		

Function		Terminates Librarian operations.
Explanation	(1)	The EXIT subcommand is used to terminate a set of Librarian operations executed by the subcommand specification.
	(2)	When executing from a subcommand file, all subcommands following after an EXIT subcommand are ignored. If the EXIT subcommand is not specified, a warning message will be displayed.
	(3)	When the EXIT subcommand is used, the immediately preceding END subcommand may be omitted. In that case the EXIT subcommand serves also as an END subcommand, causing the library file to be output before terminating the Librarian operation.

Examples	EXIT Terminates Librarian operations.

4.4.9 ABORT — Aborts Librarian operations.

Format	Name	ABORT	Option	Subcommand
			No	Yes
	Parameters	None		

Function Aborts Librarian operations.

Explanation

(1) When executing by the subcommand specification, the ABORT subcommand can be used to abort editing operations.

(2) When the ABORT subcommand is specified, the library file being edited will not be created or updated. If, however, a list file was output by a LIST subcommand before the ABORT subcommand, the list file will remain unchanged.

Examples

```
ABORT
```

Aborts Librarian operations.

4.5 List Display

4.5.1 LIST — Displays contents of a library file.

Format	Name	LIST		Option	Subcommand
				Yes	Yes
	Parameters	Option	[<List file name>]		
		Subcommand	[[<List file name>][(S)]]		

Function		Outputs a list of the contents of the library file being edited to the standard output device or to a file.
Explanation	(1)	The names of modules stored in the library file, export symbol names, and other information is output on a list. For the list format, see section 6.2, Librarian Lists.
	(2)	When no list file name is specified, the list is output to the standard output device.
	(3)	When a list file name is specified, the list is output to a file. Specify a new list file name; the list cannot be appended to an existing file. If an existing file is specified, the existing file contents will be replaced.
	(4)	When no file type is specified as part of the list file name, the type is assumed to be .lst.
	(5)	To obtain a list of export symbols designated in modules, specify the (S) parameter. If the (S) parameter is not specified, only the module names will be listed. The (S) parameter cannot be included when LIST is specified as an option.
	(6)	The LIST option or subcommand may be specified any number of times during the editing process. The library file contents at the point of specification will be listed.

(Continued on next page)

Examples –LIST

A list is output to the standard output device.

Export symbols are not shown.

LIST

A list is output to the standard output device.

Export symbols are not shown.

LISTΔlibx(S)

A list including export symbols is output to a file named libx.lst.

Section 5. Input to the Librarian

5.1 Object Module Files

Object module files output from a C compiler or assembler can be input to the Librarian and stored as modules in library files.

5.2 Relocatable Load Module Files

A relocatable load module file output from the Linkage Editor can be input and stored in a library file as one module.

5.3 Library Files

The Librarian inputs the library file it is editing. Also, modules to be stored in this library file can be input from other library files. Either specified modules can be input, or all the modules in a library file can be input at one time.

Input can be made only from library files created using this Librarian.

Section 6. Output from the Librarian

6.1 Library Files

The Librarian can combine two or more modules into a single output library file. It can also update an existing library file, or extract modules from an existing library file, and output the result in library file format.

6.2 Librarian Lists

When the LIST option or subcommand is specified, a list of the library file contents is output to the standard output device or to a file. The format of a librarian list is shown in Figure 6-1.

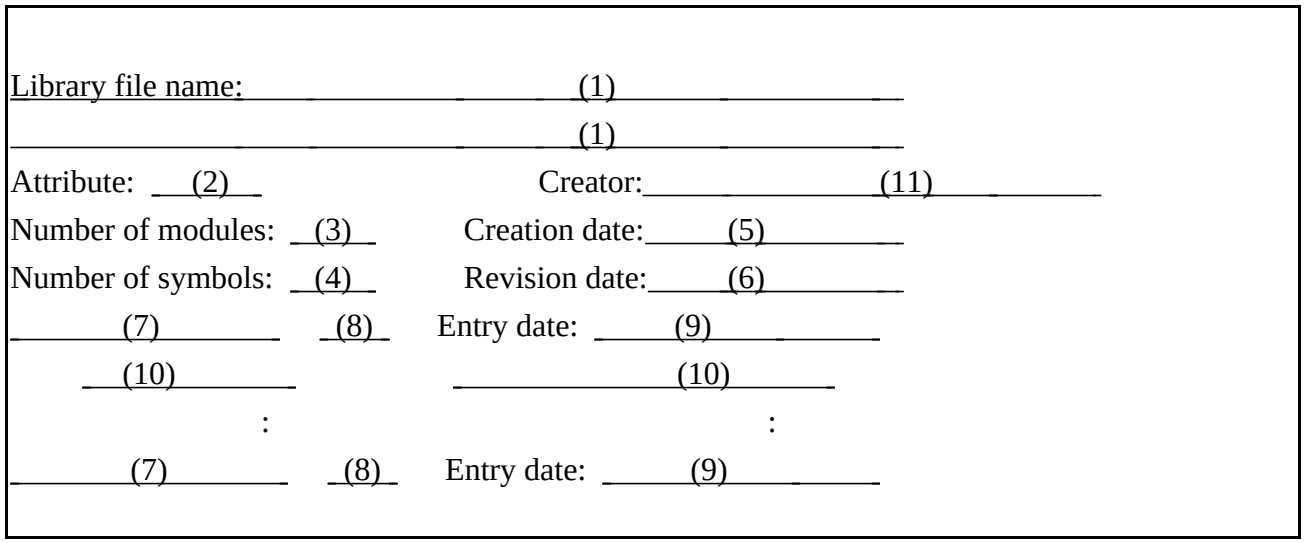


Figure 6-1. Librarian List Format

(1) Shows the library file name. If the name is too long to fit on one line it is continued to the next line. When modules are extracted from an existing library file, the list shows the contents of the existing library file.

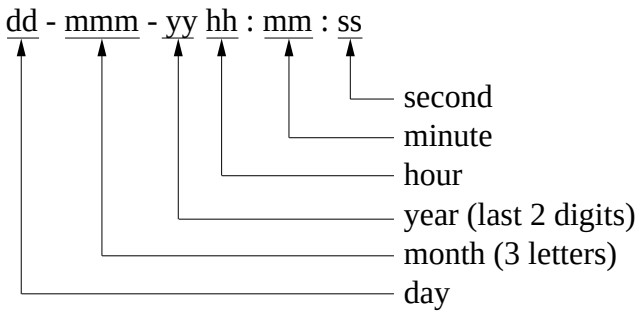
(2) Shows the library file attribute.

SYSTEM	...	System library
USER	...	User library

(3) Shows the total number of modules stored in the library file, in decimal notation.

(4) Shows the total number of externally defined symbols in the library file, in decimal notation.

(5) Shows the date and time of library file creation. This information is given in the following format.



(6) Shows the date and time of the most recent library file update. In the case of library files newly created using the CREATE option or subcommand, this is the same as the date of creation. The format is the same as for the creation date, above.

(7) Shows the names of modules stored in the library file, in alphabetical order.

(8) Shows the kind of editing operation performed on the module.

BLANK ... a module stored in an existing library file
(A) ... an added module
(R) ... a replacement module
(E) ... an extracted module

Modules deleted by means of the DELETE option or subcommand are not listed.

(9) Shows the date and time a module was stored in the library file. The format is the same as for the library file creation date and revision date.

(10) When the (S) parameter is specified with the LIST subcommand, the export symbols in each module are shown. These symbol names are listed in alphabetical order two on each line.

An example of a list when the (S) parameter is specified with the LIST subcommand is given in Figure 6-2. Figure 6-3 shows a list without the (S) specification.

MS-DOS system:

(11) Shows the name of the tool used to create the library file (Librarian model). If the name is longer than 40 characters, only the first 40 characters are shown.

```
Library file name:      clib.lib
Attribute:              USER
Number of modules:  6   Creation date:  08-Jan-90 14:18:47
Number of symbols:  6   Revision date: 01-Mar-90 19:56:33

ABS.C                       Entry date:     08-Jan-90 14:18:47
        _abs
ATOF.C                      Entry date:     08-Jan-90 14:18:47
        _atof
ATOI.C                      Entry date:     08-Jan-90 14:18:47
        _atoi
ATOL.C                      Entry date:     08-Jan-90 14:18:47
        _atol
_ALOCBUF            (A)     Entry date:     01-Mar-90 19:56:33
        _alcobuf
_DIVI               (A)     Entry date:     01-Mar-90 19:56:33
        _divi
```

Figure 6-2. Librarian List (with (S) specification)

```
Library file name:  clib.lib
Attribute:  USER
Number of modules:  6   Creation date:  08-Jan-90 14:18:47
Number of symbols:  6   Revision date: 01-Mar-90 19:56:33

ABS.C                       Entry date:     08-Jan-90 14:18:47
ATOF.C                      Entry date:     08-Jan-90 14:18:47
ATOI.C                      Entry date:     08-Jan-90 14:18:47
ATOL.C                      Entry date:     08-Jan-90 14:18:47
_ALOCBUF            (A)     Entry date:     01-Mar-90 19:56:33
_DIVI               (A)     Entry date:     01-Mar-90 19:56:33
```

Figure 6-3. Librarian List (no (S) specification)

6.3 Console Messages

The Librarian displays the following messages on the standard output device.

Opening Message: Displayed when the librarian command is input.

```
H SERIES  OBJECT  LIBRARIAN  Ver. 1.2B
Copyright (C) Hitachi, Ltd. 198X
Licensed Material of Hitachi, Ltd.
```

Normal Completion Message: Displayed when library file editing has ended normally.

```
OBJECT  LIBRARIAN  COMPLETED
```

Abort Message: Displayed when the library file editing is aborted by either an error or an ABORT subcommand.

```
OBJECT  LIBRARIAN  ABORT
```

Subcommand Prompt: Indicates that the Librarian is in subcommand input wait state during interactive execution.

```
:
```

Subcommand Continuation Symbol: Request for a continuation line, when continuation of a subcommand is specified during interactive execution.

```
-
```

Section 7. Error Messages

The Librarian outputs error messages in the following form.

** <Error number> <Error message> [(<Additional information>)]

Error number: The first digit indicates the level of the error. (xx represents the second and third digits.)

1xx : Warning	Processing of a particular module is skipped.
2xx : Error	If started by input from the command line or a subcommand file, processing is stopped. In interactive mode, processing of the subcommand is stopped when the error is detected, and a prompt is displayed for the next subcommand.
3xx : Fatal error	Processing is stopped.

A list of error messages is given below in Tables 7-1, 7-2 and 7-3, in the following format.

Error number	Error message	Additional information
Description of error		
Corrective action, etc.		

Note: Additional information includes the name of the file in which the error occurred, or the module name or symbol name. In the list of errors, --- means that no additional information is given.

Table 7-1. List of Warning Messages

101	DUPLICATE MODULE	Module name	
	An attempt was made to add a module already stored in the library file.		
	Processing of the module is skipped.		
102	DUPLICATE SYMBOL	Module name **	Symbol name
	An attempt was made to add an export symbol already present in the library file.		
	Processing of the module is skipped.		
103	MODULE NAME TOO LONG - ALLOWED UP TO 32	Module name	
	A module name of more than 32 characters was specified.		
	The name is valid up to the 32nd character. The rest is ignored.		
104	EXIT SUBCOMMAND NOT FOUND - ASSUMED	---	
	No EXIT subcommand was specified.		
	Processing continues as though an EXIT subcommand had been specified.		

Table 7-2. List of Error Messages

201	INVALID SUBCOMMAND/OPTION	---
	The option or subcommand specified is invalid in this context.	
	Specify a valid option or subcommand.	
202	SYNTAX ERROR	---
	Syntax of the specified option or subcommand is incorrect.	
	Check the syntax and re-specify the option or subcommand.	
203	SUBCOMMAND LINE LENGTH	---
	TOO LONG Length of the subcommand entry exceeds 128 characters.	
	Re-specify, keeping the length within 128 characters.	
204	CONFLICTING SUBCOMMAND	---
	Subcommands are specified in the wrong order, or an illegal combination of subcommands is specified.	
	Check the order of subcommands and re-specify.	
205	ILLEGAL FILE NAME	---
	The specified file name is not valid.	
	Specify a correct file name.	
206	ILLEGAL MODULE NAME	---
	The specified module name is not valid.	
	Specify a correct module name.	
207	MODULE NOT FOUND	Module name
	The specified module cannot be found.	
	Check the name of the module, then re-specify.	

(Continued on next page)

Table 7-2. List of Error Messages (cont)

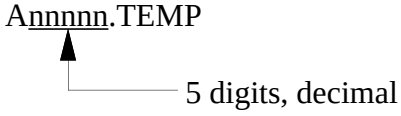
208	MISSING OUTPUT FILE NAME	---
	No output file was specified with an EXTRACT option or subcommand.	
	Use the OUTPUT option or subcommand to specify an output file.	
209	TOO MANY INPUT FILES	---
	More than 12 input files were specified for input at the same time.	
	First output the library file, then re-input the library file and input the remaining files.	
210	TOO MANY MODULES	---
	The number of modules exceeds the allowable number.	
	No more modules can be stored in the library file now being created or edited. Store any additional modules in a separate library file.	
211	TOO MANY SYMBOLS	---
	The number of symbols exceeds the allowable number.	
	The library file now being created or edited cannot contain any more symbols. Modules with additional symbols must be stored in a separate library file.	
212	ILLEGAL FILE FORMAT	---
	The specified file format is incorrect.	
	Check the file contents and re-execute.	
213	MEMORY OVERFLOW	---
	There is no space remaining in the Librarian's usable memory.	
	Obtain additional memory and re-execute.	
214	FILE NOT FOUND	File name
	The specified file cannot be found.	
	Check the directory and the specified file name, then re-specify.	

(Continued on next page)

Table 7-3. List of Fatal Error Messages

301	INVALID COMMAND PARAMETER	---
	An improper command parameter was specified.	
	Check the command parameters and re-execute.	
302	CONFLICTING OPTION	---
	There is a contradiction among different options specified.	
	Check the order of option specification, then re-specify.	
303	CANNOT OPEN FILE	File name
	File cannot be opened, or the CREATE or OUTPUT option or subcommand specified an already existing file.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware error. Check the problem, then re-execute. If an existing file was specified by the CREATE or OUTPUT option or subcommand, delete the existing file, then re-execute.	
304	CANNOT INPUT FILE	File name
	File cannot be input.	
	Check the specified file name. If the file name is correct, there may be a disk hardware error. Check the problem, then re-execute.	
305	CANNOT OUTPUT FILE	File name
	File cannot be output.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware error. Check the problem, then re-execute.	
306	CANNOT CLOSE FILE	File name
	File cannot be closed.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware error. Check the problem, then re-execute.	

Note: The Librarian uses temporary files with names in the format shown below. These temporary file names may appear as additional information in error messages.



Section 8. Restirictions

Restriction on the Librarian are shown in Table 8-1. If the numerical restrictions are exceeded, Librarian operations will not execute correctly.

Table 8-1. Restrictions on Librarian Processing

No.	Item	Limits	Remarks
1	The number of modules that can be stored in a library file	Max. 32,767	Assumes that the system on which Librarian runs has adequate memory.
2	The number of symbols that can be present in a library file	Max. 65,535	
3	The number of input files	Max.12	Total number of files specified by LIBRARY, ADD, or REPLACE not including subcommand files.
4	The number of modules that can be specified in a library file	Max. 10	When specifying a library file with ADD or REPLACE
5	Length of command line	Depends on OS	
6	Length of option or subcommand	Max. 128 characters	Not including RET
7	Length of file name	Max. 128 characters	Includes default file-type characters. File name format depends on OS.
8	Length of module name	Max. 32 characters	
9	Length of symbol name	Max. 32 characters	
10	Input file formats	• Object module file output by assembler or C compiler. • Relocatable load module file. • Library file created using this Librarian.	

Appendix A Examples of Use of Librarian

A.1 Librarian Execution by Command Line

```
% lbrΔ-CREATE=func-ADD=abs,mod,sqrt,exp,log 3
          1          2
```

...(1) Creation

```
% lbrΔfunc-ADD=sin,cos-DELETE=abs,mod-LIST 3
       3      4          5          6
```

...(2) Editing

```
% lbrΔfunc-EXTRACT=sqrt,exp-OUTPUT=newfnc 3
       7          8             9
```

...(3) Extraction

1. The CREATE option at the beginning of the option line is specified to create a new library file.
2. The file names for the modules to be entered are specified using the ADD option.
3. The name of the library file to be edited is specified.
4. The file names for modules to be added to the existing library file are specified using the ADD option.
5. The names of the modules to be deleted from the existing library file are specified using the DELETE option.
6. The LIST option is specified to confirm the editing results.
7. An existing library file from which modules are to be extracted is specified.
8. The names of the modules to be extracted are specified using the EXTRACT option.
9. The name of a new library file to which the extracted modules are to be output is specified using the OUTPUT option.

This process is illustrated in Figure A-1.

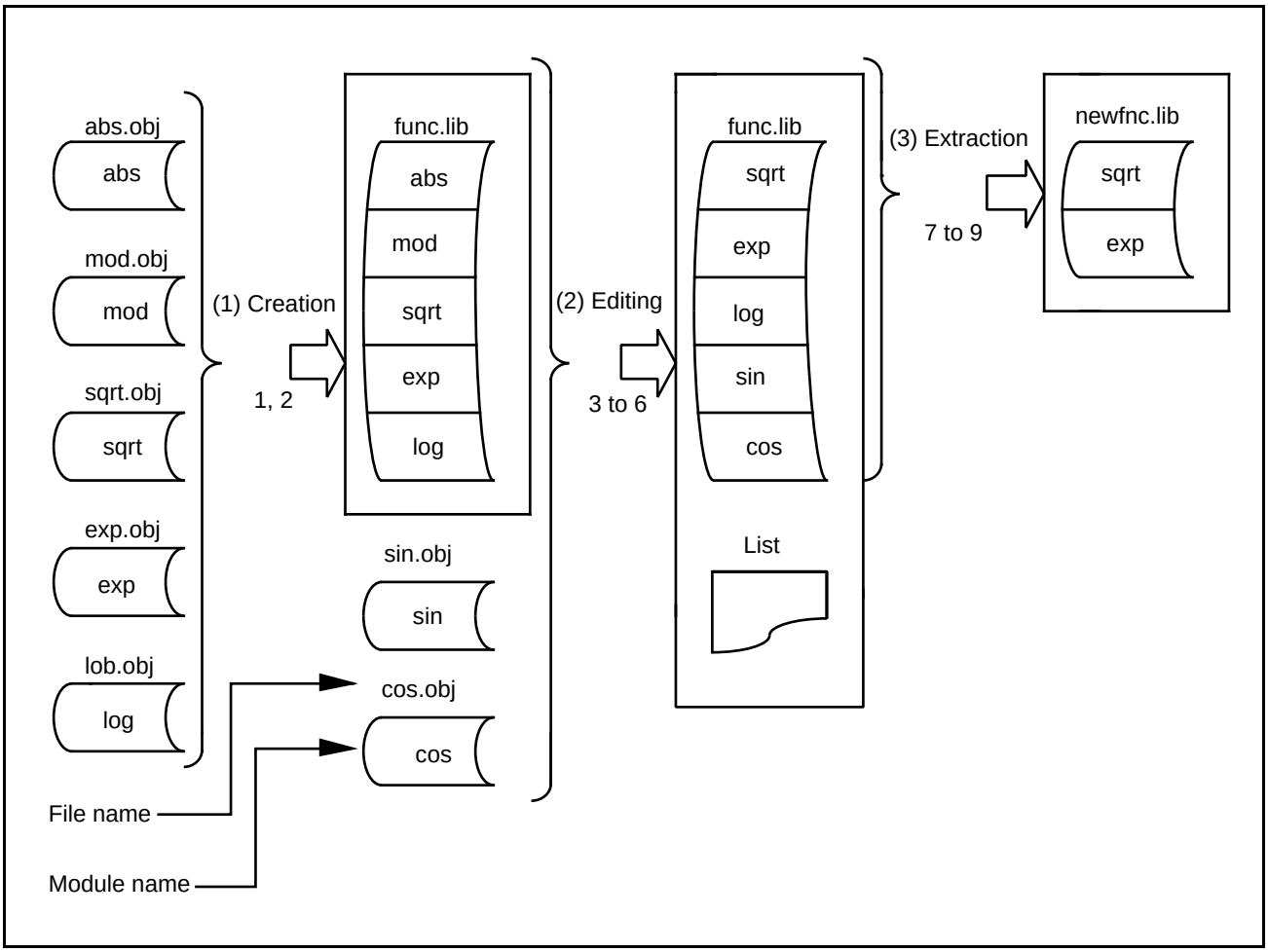


Figure A-1. Results of Librarian Execution by Command Line

A.2 Librarian Execution by Subcommands

```
%  lbr ↵                                          ...1
:  CREATE␣func ↵                                  ...2
:  ADD␣sqrt,exp,log,sin,cos ↵                     ...3
:  END ↵                                          ...4
:  LIBRARY␣func ↵                                 ...5
:  REPLACE␣sin.new,cos.new,tan.new ↵              ...6
:  END ↵                                          ...7
:  LIBRARY␣func ↵                                 ...8
:  LIST ↵                                         ...9
:  EXTRACT␣sqrt,exp ↵                             ...10
:  OUTPUT␣newfnc ↵                                ...11
:  END ↵                                          ...12
:  EXIT ↵                                         ...13
```

(1) Creation — (2) Editing — (3) Extraction

1. The Librarian is started.
2. The CREATE subcommand at the beginning of the option line is specified in order to create a new library file.
3. The file names of modules to be loaded are specified using the ADD subcommand.
4. The END subcommand is specified to terminate the creation process.
5. The name of the library file to be edited is specified.
6. Modules in the existing library file are replaced, using the REPLACE subcommand. The file names of the modules to be replaced is specified.
7. The END subcommand is specified to terminate the editing process.
8. An existing library file is designated for extraction of modules.
9. The LIST subcommand is specified to confirm the contents of the existing library file.
10. The names of the modules to be extracted are specified using the EXTRACT subcommand.
11. The name of a new library file to which the extracted modules are to be output is specified using the OUTPUT subcommand.
12. The END subcommand is specified to terminate the extraction process.
13. The EXIT subcommand is specified to terminate the Librarian program.

This process is illustrated in Figure A-2.

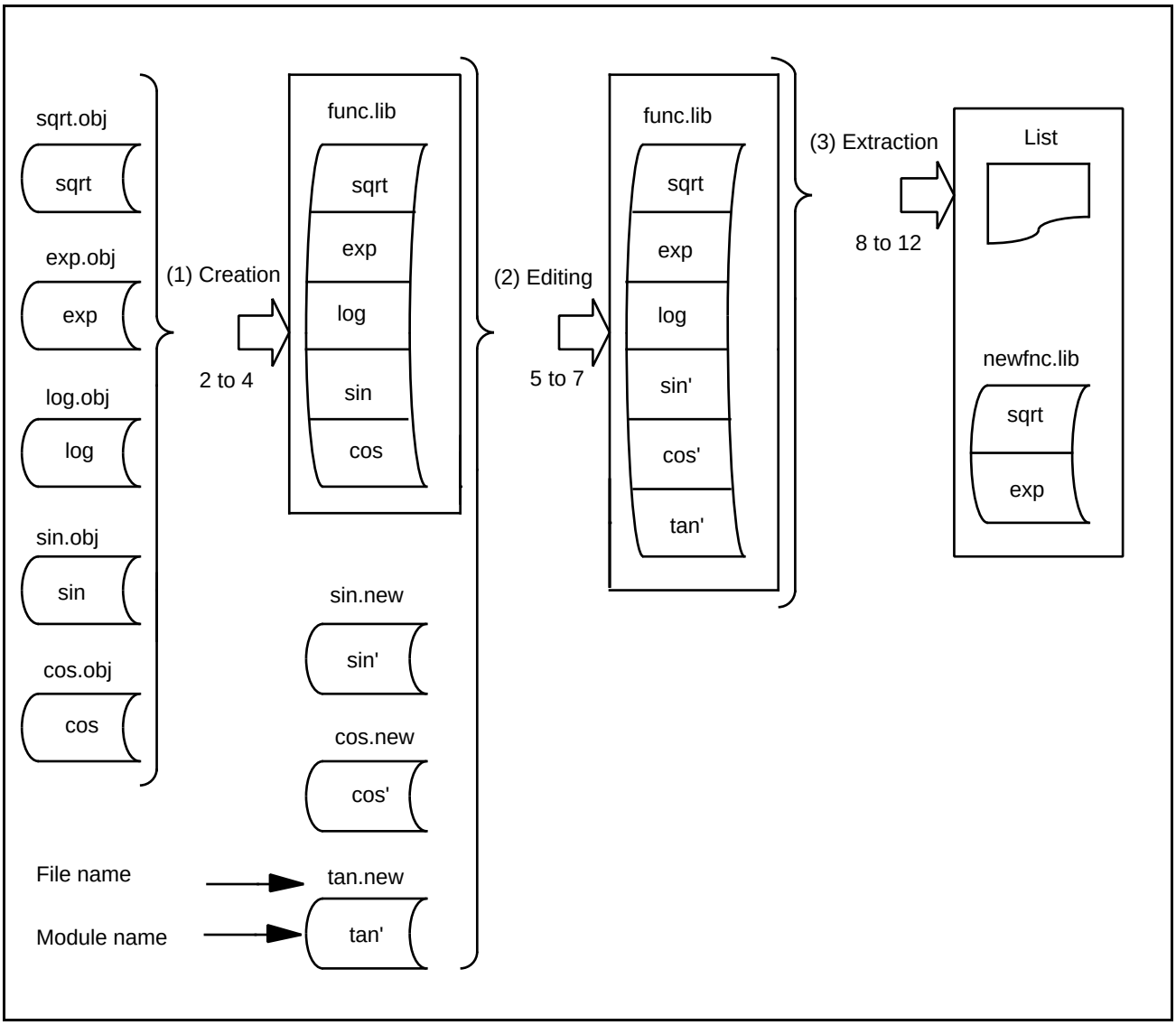


Figure A-2. Results of Librarian Execution by Subcommand

Index