

# Publications Bulletin

2200.005-0001

4/2/73

Release 2 Update Package for:

**MRX/OS Program Library Services  
Reference Manual  
2200.005**

This bulletin advises of changes that have occurred to the **Program Library Services Reference Manual** since the **March 1973** edition was issued. New and replacement pages are provided where required.

<u>Pages</u>	<u>Action</u>
v and vi	Replace
2-7 thru 2-44	Replace
3-1 thru 3-4	Replace
3-19 thru 3-28	Replace
4-3 thru 4-6	Replace
A-3 and A-4	Replace
A-7 and A-8	Replace
Appendix B	Replace
Index	Replace

*Technical* changes to text, tables, and figures are marked with a vertical bar in the outer margin.

Pages containing *non-technical* changes (page layout, spelling corrections) are indicated by a bar opposite the page number.

Please file this bulletin with the publication to retain a record of changes.

Sequence Number: M105

MEMMOREX

Computer System  
Products

# TABLE OF CONTENTS

Section		Page
1	INTRODUCTION	1-1
	Coding	1-1
	Compilation	1-1
	Linkage Editing	1-1
	Library Processing	1-2
	Loading	1-2
2	LIBRARIAN	2-1
	Introduction	2-1
	Library Description	2-1
	Library Structure	2-1
	Data Structure	2-2
	Library Utility Program (LIBUTIL)	2-4
	Control Language Requirements	2-4
	Data Separator Statement	2-4
	Programming Considerations	2-5
	Keyword-Operand Descriptions	2-6
	Function Keyword (COMMAND or COM)	2-6
	Input Library (ILIB)	2-6
	Output Library (OLIB)	2-6
	Member Name (MEM)	2-7
	Member Type (MTYPE)	2-8
	Member Selection (SELECT)	2-9
	Listing (LIST)	2-9
	Listing Title (TITLE)	2-9
	Initial Page Number (INITPG)	2-10
	Page Size (PGSIZE)	2-10
	Line Spacing (SPACE)	2-10
	Version Number (VERSION)	2-10
	Sequence Field Definition (SEQPOS)	2-10
	Sequence Renumbering (NEWSEQ)	2-11
	Sequence Checking (SEQCHK)	2-11
	Dump Output Format (MODE)	2-11
	Update Mode (UMODE)	2-12
	Primary Input File (IFIL)	2-12
	Output Data File (OFIL)	2-12
	Patch Library (ULIB)	2-12
	Work Library (WLIB)	2-13
	Command Descriptions	2-13
	Print Table of Contents (PTOC)	2-13
	Sample PTOC Listing	2-16

## TABLE OF CONTENTS (Continued)

Section	Page
2 (Cont)	<ul style="list-style-type: none"> <li>Copy Library Member (COPY) 2-19</li> <li>Delete Library Member (DELETE) 2-21</li> <li>Compress Library (PACK) 2-23</li> <li>Assign New Member Name (RENAME) 2-26</li> <li>Punch Encoded Member (DUMP) 2-29</li> <li>Load Dumped Member (LOAD) 2-32</li> <li>Modify Load Member (PATCH) 2-34                             <ul style="list-style-type: none"> <li>Patching Relocatable Load Modules 2-35</li> <li>PATCH Examples 2-36</li> </ul> </li> <li>Print Symbolic Member (PRINT) 2-38</li> <li>Punch Symbolic Member (PUNCH) 2-39</li> <li>Create or Modify Symbolic Member (UPDATE) 2-41                             <ul style="list-style-type: none"> <li>Pointer Directives 2-43                                     <ul style="list-style-type: none"> <li>Pointer by Relative Record Number 2-43</li> </ul> </li> <li>Copy Directive 2-47</li> </ul> </li> </ul>
3	<ul style="list-style-type: none"> <li>LINKAGE EDITOR 3-1</li> <li>Functional Description 3-1                             <ul style="list-style-type: none"> <li>Module Linkage and Editing 3-1</li> <li>Additional Input Sources 3-1</li> <li>Storage Reservation 3-2</li> <li>Overlay Program Creation 3-2</li> <li>Special Processing and Error Diagnosis 3-2</li> <li>Load Module Attribute Assignment 3-2</li> </ul> </li> <li>Input Structure 3-3                             <ul style="list-style-type: none"> <li>Basic Input 3-3</li> <li>Secondary Input 3-3                                     <ul style="list-style-type: none"> <li>Library Search Domain 3-4</li> <li>Entry Point Search Domain 3-4</li> </ul> </li> </ul> </li> <li>Object Modules 3-5</li> <li>Language Processor Relationships 3-5                             <ul style="list-style-type: none"> <li>Assembler 3-6</li> <li>COBOL 3-6</li> <li>FORTRAN 3-6</li> <li>RPG II 3-7</li> </ul> </li> <li>Output 3-7                             <ul style="list-style-type: none"> <li>Absolute Load Modules 3-8</li> <li>Relocatable Load Modules 3-8</li> <li>Load Module Creation 3-8</li> </ul> </li> <li>Control Language Statement Descriptions 3-9                             <ul style="list-style-type: none"> <li>File Definition 3-9                                     <ul style="list-style-type: none"> <li>List File 3-9</li> <li>Input File 3-9</li> </ul> </li> </ul> </li> </ul>

The ident operand must be the same as that specified by the ID keyword of the //DEFINE statement for that library. SYSIN and SYSOUT are illegal operands. If omitted, the standard default, OUTPUT, is assumed.

The type operand is optional and can be one of the following:

ALL – Specifies that any type of member can be included on the library. Block size must be 256 bytes. This is the default.

ENC – Specifies that only object modules or load modules are on the library. Block size must be 256 bytes.

SYM – Specifies that only source, macro, or procedure members are on the library. Minimum block size is 84 bytes; maximum is 512 bytes. Block size greater than 128 bytes makes procedure members inaccessible to Control Language Services.

OLIB applies to the COPY, DELETE, LOAD, PACK, RENAME, and UPDATE commands.

#### **MEMBER NAME (MEM)**

The MEM keyword is used to specify member names and types, and protection of an already existing member of the same name and type, for all LIBUTIL functions that involve named members. In addition, MEM can be used in conjunction with the SELECT keyword to include or exclude a given set of members when an entire library is being processed. MEM is specified once for each member; the number of MEM keywords allowed for each command is summarized in Table 2-1.

The MEM keyword includes from one to five operands, as follows:

MEM=([input member name] [,type] [,output member name] [,type] [,P])

Input member name is a 1- to 8-character alphanumeric field that specifies the member identification as listed in the library catalog. This operand is required when the MEM keyword is used with the COPY, DELETE, DUMP, PACK, PATCH, PRINT, PUNCH, and RENAME commands. It is optional for the UPDATE commands. When it is omitted with the UPDATE command, a new member is created from the data in the accompanying directive file.

Output member name is a 1- to 8-character alphanumeric field that specifies the new name under which the member is to be listed in the library catalog. It is required for the RENAME and UPDATE commands and is optional for COPY and PACK. For RENAME, the catalog entry of the input member name will be marked deleted and replaced by the output member name, unless protection is specified. For COPY, PACK, and UPDATE, any member on the output file having the same name as the specified output member name has its catalog entry deleted, provided it is of the same type, unless protection is specified. For COPY and PACK, omission of the output member name implies that the input member name is used as the output member name.

Member type may be specified with input and output member names when they are used. The valid types are as follows:

SRC	Source member
PRO	Control Language procedure member
MAC	Unassembled macro member
OBJ	Object member
REL	Relocatable load member
ABS	Absolute load member

When this operand is omitted, the MTYPE keyword operand must be supplied. However, the value specified for member type with the MEM keyword overrides the value specified by MTYPE.

Example:

In this example, MTYPE identifies ALT1, ALT2, and ALT3 as source type (SRC). The final line is not overridden by MTYPE, since macro type (MAC) is specified for both ALT4 and ALT41.

```
//PAR MTYPE=SRC,COMMAND=COPY,  
//PAR MEM=ALT1,MEM=ALT2,MEM=ALT3,  
//PAR MEM=(ALT4,MAC,ALT41,MAC,P)
```

Protection of an existing member on a library can be specified by including the character P as the last operand for MEM. When P is included, the presence on the output library of a member with a name and type identical to that specified by the output member name operand results in program abort. When P is omitted, the new member will be added unconditionally, and any identically named member of the same type will be marked as deleted.

Examples of MEM keyword-operand configurations are shown below:

```
MEM=MYNAME  
MEM=(OLDNAME,SRC)  
MEM=(OLDNAME,,NEWNAME)  
MEM=(,,NAMEONE,MAC,P)  
MEM=(OLD,PRO,NEW,PRO)  
MEM=(ALTER,OBJ,,P)
```

#### MEMBER TYPE (MTYPE)

This keyword-operand specifies a default for the member type wherever it has not been specified in the MEM parameter. The MTYPE keyword applies to all commands except PTOC and LOAD. MTYPE must be supplied if member type is omitted after any explicit member name specification, and applies to all member names for which a member type is

not specified. However, where member type is given with the member name operand, the MTYPE operand has no effect.

The operands for the MTYPE keyword are the same as those specified for the member type operands of the MEM keyword.

#### **MEMBER SELECTION (SELECT)**

The SELECT keyword specifies that either an inclusive or an exclusive operation is requested. There are two operands for this keyword: I (inclusive) which specifies that only the members named are to be included, and E (exclusive) which means only the named members are excluded. When this keyword is not coded, the default depends upon the use of the MEM keyword-operand. If the MEM keyword has been specified, the default is I (inclusive). However, if the MEM parameter has not been specified, the default is E (exclusive); that is, no members are excluded. This keyword applies only to COPY and PACK commands.

#### **LISTING (LIST)**

The LIST keyword-operand specifies whether a listing of messages, updated elements, and other information is to be performed as a result of LIBUTIL processing. There are two operands for this keyword, YES and NO. The default is YES.

YES specifies that a complete listing is to be produced. NO specifies that a partial listing will be produced including the title line and a summary of parameters received, with a function complete message and/or coded error messages. A Control Language //DEFINE statement must be included in the step, containing ID=LIST as its file identifier and DEV=PRT for device specification regardless of how LIST is coded.

This keyword is applicable to all LIBUTIL functions. LIST=NO is illegal for PTOC and PRINT commands.

#### **LISTING TITLE (TITLE)**

The TITLE keyword-operand specifies a title line, given as a literal string constant, which is to appear on each page of the output listing preceding all detail lines. The literal string must be enclosed in apostrophes, and must be contained on one card not to exceed 50 characters.

When this parameter is not coded, a line containing the system date and time and the LIBUTIL function appears as a header. When the parameter is coded, the system page header is not printed. This keyword applies to all of the LIBUTIL commands.

Example:

```
TITLE='PROCEDURES STORED ON LIBRARY 17'
```

#### **INITIAL PAGE NUMBER (INITPG)**

This keyword-operand specifies a decimal value as the initial page number of the output listing designated with the LIST keyword. The INITPG operand is a 1- to 3-digit value in the range of 1 to 999. If INITPG is not specified, 1 is the default. INITPG can be designated for any command except PTOC.

#### **PAGE SIZE (PGSIZE)**

The PGSIZE keyword defines the maximum number of lines to be printed per page on the output listing. All header, title, and blank lines are included in the count. The PGSIZE operand applies to all of the LIBUTIL commands.

This operand is a 1- to 2-digit decimal value in the range 1 to 99. When this parameter is not specified, a default value of 60 is assumed.

#### **LINE SPACING (SPACE)**

The SPACE keyword specifies the line spacing for the output listing specified ID=LIST. The operand can be 1, 2, or 3, meaning single, double, or triple spaced detail lines. Single spacing is provided when SPACE is not specified. This keyword applies to all LIBUTIL commands except PTOC.

#### **VERSION NUMBER (VERSION)**

This keyword specifies the numeric value that is to be stored in the library catalog entry as the version identifier for the output member. The version identifier is used only for convenience in identifying modules from a listing and does not modify the file identifier.

The operand of the VERSION keyword is a four character numeric field. This keyword applies only to the RENAME and UPDATE commands. When this keyword is not coded, the default value used is a 0000 character field for initial creation of the member. For all other uses of UPDATE and for the RENAME command, the default is the value already in the version identifier of the library catalog member entry. The version number for members involved in the COPY, PACK, and PATCH commands reverts to zero.

#### **SEQUENCE FIELD DEFINITION (SEQPOS)**

The SEQPOS keyword-operand defines the starting position and length of the sequence field in the card records of the library member being processed. This keyword includes two operands, both of which must be specified whenever SEQPOS is used. The format is as follows:

SEQPOS=(n,m)

Operand n is a two-digit decimal value that specifies the starting position of the sequence field in the card record. Operand m is a one-digit value from 1 to 8 specifying the length of the sequence field. The starting position (operand n) plus the length of the sequence field (operand m) minus 1 must not be coded such that the sequence field extends beyond the record limit.

This keyword applies to the PUNCH and UPDATE command only. If not specified, the default is SEQPOS=(73,8). The SEQPOS operand is ignored unless NEWSEQ or SEQCHK is coded.

#### **SEQUENCE RENUMBERING (NEWSEQ)**

The NEWSEQ keyword-operand specifies a sequence field renumbering operation as part of LIBUTIL processing. Renumbering generates sequential numbers in the positions defined as the sequence field by the SEQPOS parameter. The format is as follows:

$$\text{NEWSEQ} = \left\{ \begin{array}{l} \text{YES} \\ (n,m) \\ \text{NO} \end{array} \right\}$$

The operand YES specifies that renumbering will occur with the values 100 for the initial number in the sequence and 100 as the increment for each succeeding record. In the format (n,m), operand n specifies an initial value of the sequence counter in the range 1 to 9999, while operand m supplies the increment for each succeeding record in the range 1 to 9999. Operand NO specifies no renumbering is to occur, and is the default. NEWSEQ applies only to the UPDATE and PUNCH commands.

#### **SEQUENCE CHECKING (SEQCHK)**

The SEQCHK keyword-operand specifies a sequence check which verifies the ascending order of the output sequence field. The operands are YES and NO. When YES is coded, the sequence check will be performed. Any discrepancies will be flagged on the output listing. This check results in program abort if a sequencing violation is found.

When NO is coded, the sequence check will be omitted. The default is NO. This keyword applies only to the UPDATE command.

#### **DUMP OUTPUT FORMAT (MODE)**

This keyword applies to the DUMP command only. It specifies the format in which a load or object member of a library is to be dumped to punched cards. There are two operands for this keyword: R and M. MODE=R specifies that the object member is to be dumped in reloadable format, which allows the member to be reloaded with the LIBUTIL LOAD command. MODE=M specifies that the object member is to be dumped in machine loadable format. M is valid only for absolute load modules. (It allows users to punch out stand alone programs which can be reset loaded.) Members dumped in this mode cannot be reloaded with the LOAD command. The default is reloadable format.



#### **UPDATE MODE (UMODE)**

The UMODE keyword specifies the update method to be used, and has two operands, SEQ and REL. UMODE=SEQ designates that the sequence numbers on the symbolic statements are used in the UPDATE. UMODE=REL specifies that the relative record numbers associated with the file as shown on the previous UPDATE or PRINT listing are used. The default is REL. UMODE applies only to the UPDATE command and is explained in detail in conjunction with that command.

#### **PRIMARY INPUT FILE (IFIL)**

The IFIL keyword-operand designates the primary input data file to be used in LIBUTIL processing by the LOAD, PATCH, or UPDATE commands only. It does not apply to the other LIBUTIL commands. The operand is a 1- to 8-character alphanumeric value and must be specified as the ID of a //DEFINE statement within the step.

For the LOAD function, the named file contains the dumped members to be reloaded. In the PATCH function, the file contains the object patches and directives for the named member. With UPDATE, the file contains the source changes and/or directives to the named member. When IFIL is not specified, the default SEQIN is used; the //DEFINE statement must still be included.

#### **OUTPUT DATA FILE (OFIL)**

This keyword specifies the output data file to be used for the DUMP and PUNCH commands only. The operand is a 1- to 8-character alphanumeric value that must match the operand of the ID keyword of a //DEFINE Control Language statement in the step.

The named file will receive the encoded member being dumped, or the card images of the symbolic member being punched. When OFIL is not specified, the default SEQOUT is used.

In this event, ID=SEQOUT must be specified on the //DEFINE statement.

#### **PATCH LIBRARY (ULIB)**

This keyword-operand specifies the library containing the member to be modified by the PATCH function. This keyword does not apply to any other LIBUTIL functions. The operand is a 1- to 8-character alphanumeric value, and must be specified as the operand of the ID keyword on a //DEFINE Control Language statement within the step. When this keyword is omitted, the default UPDATE is used. A //DEFINE statement must be included for the update library, whether or not the default is used.

## **WORK LIBRARY (WLIB)**

The WLIB keyword specifies the primary library work file used by the LIBUTIL PACK function only. This keyword does not apply to any of the other LIBUTIL commands. The operand is a 1- to 8-character alphanumeric value and must be designated as the ID operand of a //DEFINE statement within the step. When WLIB is not specified for the PACK command, the default WORK will be used. The //DEFINE statement for this file must still be included in the step whether or not the default is used.

## **COMMAND DESCRIPTIONS**

All LIBUTIL commands perform general Librarian Utility functions for the programmer. They may, however, optionally be directed to handle more detailed operations by specifying particular keywords in the request for the function. For example, deletion-flagging can be caused directly or indirectly by the DELETE, UPDATE, LOAD, COPY, and RENAME commands. The following paragraphs discuss each LIBUTIL command, its functions and capabilities, the applicable keywords, and examples of use. Commands have been grouped more or less by the library types to which they apply. The first group, consisting of PTOC, COPY, DELETE, PACK, and RENAME, apply to all types of libraries; the second group, consisting of DUMP, LOAD, and PATCH, apply to the encoded type of library; and the third group, consisting of PRINT, PUNCH, and UPDATE, apply to symbolic type libraries. Librarian error codes are listed in Appendix B.

Table 2-1 lists all of the keywords used to specify the various LIBUTIL operands and the functions to which they apply. Use of a keyword with a command to which it does not apply results in the Librarian issuing a warning code and continuing. No error action is taken. Once the warning code is issued, the keyword is ignored.

The keywords that apply to all of the LIBUTIL functions are: COMMAND, LIST, PGSIZE, and TITLE.

## **PRINT TABLE OF CONTENTS (PTOC)**

The PTOC function displays the contents of the named library directory on a print file list. The list shows names and characteristics of each member in the library. Members will be displayed in chronological order of creation date and time. Deleted members of the file will always be included in a PTOC listing and will be marked deleted. A LIST file must be specified for this function, either by LIST=YES or by default. The listing will be single spaced.

The content of the //PAR statement used for the PTOC function is:

```
COMMAND=PTOC
[,LIB=library identifier]
[,LIST=YES]
[,PGSIZE=lines per page]
[,TITLE='literal string']
```

Table 2-1. Summary Table of LIBUTIL. Keywords by Command\*

Keyword <sup>①</sup>	Default	Entries by Command										
COMMAND or COM	None <sup>①</sup>	COPY	DELETE	DUMP	LOAD	PACK	PATCH	PRINT	PTOC	PUNCH	RENAME	UPDATE
IFIL	SEQIN				O		O					O
ILIB	INPUT	O		O				O	O	O		O
INITPG	1	O	O	O	O	O	O	O		O	O	O
LIST	YES	O	O	O	O	O	O	O <sup>②</sup>	O <sup>②</sup>	O	O	O
MEM	None	O <sup>③</sup>	R <sup>④</sup>	R <sup>④</sup>		O <sup>③</sup>	R <sup>④</sup>	R <sup>④</sup>		R <sup>④</sup>	R <sup>⑤</sup>	R <sup>⑥</sup>
MODE	R			O								
MTYPE	None	O	O	O		O	O	O		O	O	O
NEWSEQ	NO									O		O
OFIL	SEQOUT			O						O		
OLIB	OUTPUT	O	O		O	O						O
PGSIZE	60	O	O	O	O	O	O	O	O	O	O	O
SELECT	I/E <sup>⑦</sup>	O				O						
SEQCHK	NO											O
SEQPOS	(73,8)									O		O
SPACE	1 (Single space)	O	O	O	O	O	O	O		O	O	O
TITLE	System date, time, command, and page number	O	O	O	O	O	O	O	O	O	O	O
ULIB	UPDATE						O					
UMODE	REL											O
VERSION	0000 <sup>⑧</sup>										O	O
WLIB	WORK					O						

Key: R Required Keyword  
O Optional Keyword  
blank Keyword does not apply to the command

\*Circled numbers in the table refer to the following notes.

NOTES

- ① Keywords and COMMAND or COM operands must be spelled as they appear in this table.
- ② LIST=YES required, either specified or accepted as default.
- ③ Optional keyword. When coded, input-member name must always be included. Output-member name may be omitted. If member type is omitted, MTYPE must be coded. Protection is always optional. Up to ten occurrences of MEM are allowed.
- ④ Required keyword. Input-member-name must be coded. Member type may be coded; if omitted, MTYPE must be coded. Output-member-name and type do not apply. Protection is always optional. Up to ten occurrences of MEM are allowed except for PATCH, which allows only one.

NOTES (Continued)

- ⑤ Required keyword. Input- and output-member-names must be coded. Member types may be coded; if omitted, MTYPE must be coded. Protection always optional. Up to ten occurrences of MEM are allowed.
- ⑥ Required keyword. Input-member-name and type omitted if module is being created. Output-member-name must be coded. Output-member type may be coded; if omitted MTYPE must be coded. Protection is always optional. Multiple occurrences of MEM are not allowed.
- ⑦ Default is I (inclusive) includes members named when MEM is coded; E (exclusive) excludes members named (none) when MEM is not coded.
- ⑧ Default to 0000 (zeros) applies only when member is first created. Default for all other cases of UPDATE and for RENAME is to the value already in the version identifier field of the library directory member entry.

The default values listed in Table 2-2 should be used whenever possible for the PTOC function.

Table 2-2. Default Values for LIBUTIL PTOC Function

Keyword	Default
ILIB	INPUT
LIST	YES (LIST=NO is illegal for this function)
PGSIZE	60
TITLE	System header line

The following are examples of //PAR statements that request the LIBUTIL PTOC function:

Example 1:

In this example the table of contents of the library specified by ID=INPUT on the //DEF statement in the step will be printed.

```
//PAR COMMAND=PTOC
```

Example 2:

In this example the table of contents of the library specified by ID=OWNLIB will be printed 50 lines per page.

```
//PAR COMMAND=PTOC,PGSIZE=50,ILIB=OWNLIB
```

The following examples show the Control Language statements for a job step which uses the PTOC function:

**Example 3:**

In this example the table of contents of the library ORDENT7 will be listed.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=PTOC
//DEF  ID=INPUT,FIL=ORDENT7,STA=(P,I)
//DEF  ID=LIST,DEV=PRINTER
//EOJ
```

**Example 4:**

In this example the table of contents of the library ORDENT71 will be displayed. The listing will have the title ORDER ENTRY LIBRARY 7 printed at the top of each page. The ID listed as OUTPUT might be specified when the PTOC is an additional function in the same step that has just created a library using another LIBUTIL function.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=PTOC,ID=OUTPUT,
//PAR  TITLE='ORDER ENTRY LIBRARY 7'
//DEF  ID=LIST,DEV=PRINTER
//DEF  ID=OUTPUT,FIL=ORDENT71,STA=(P,I)
//EOJ
```

**Sample PTOC Listing**

The listing produced by the PTOC function begins with a system header line, with a title embedded in it if one is specified, followed by a line showing the library name and type. A sample appears below.

```
PTOC FUNCTION: DATE=73027  TIME=205423. (up to 50 character title) PAGE:001
FILE LABEL: $OSRSDNTLIB      /ALL
```

- DATE is the Julian date (year and day) used by the operating system.
- TIME is the system time in the format hhmmss for hours, minutes, and seconds.
- FILE LABEL is the name of the library specified as the FIL operand on the //DEF card specifying the input library for the function.
- Library type (ALL in the sample) follows the slash.

Then follows a string of equal signs, followed by a line of column headers as shown below.

=====  
MEMBER NAME TYPE VERSION YR/DAY HH MM SS USER DATA SUB-DIV TOP OF SUB-DIV

- MEMBER NAME is the cataloged name of the member.
- TYPE is one of the six valid types described with the MEM keyword operand.
- VERSION is the version number specified by the user or supplied by the Librarian as specified for the VERSION keyword-operand.
- YR/DAY and HH MM SS are the creation date and time when the member was entered on the library.
- USER DATA consists of the user extension words (bytes 32-41) of the member definition block for load modules (Appendix A).
- SUB-DIV refers to the data subdivisions of load and object modules specified in the member definition block (Appendix A). The numbers of the subdivisions correspond to the order in which their block numbers appear in the member definition block.

SUB-DIV 1	Bytes 20-23
SUB-DIV 2	Bytes 24-27
SUB-DIV 3	Bytes 28-31
SUB-DIV 4	Bytes 32-35 of the member definition block for object modules only

- TOP OF SUB-DIV specifies the beginning library block number of the subdivision in decimal notation. Where the block number is zero, no subdivision is present. Although the length of the last subdivision of the last member on the PTOC listing is not shown, the programmer can obtain the approximate number of available blocks remaining on the library by subtracting the last subdivision block number from the total blocks allocated to the library.

Figure 2-2 is an example of a PTOC listing showing absolute load members and object members. The letter D preceding a member name indicates that the catalog entry for the member has been marked deleted.

PTOC FUNCTION: DATE=73027  
 FILE LABEL: \$OSRSDNTLIB

TIME=205423.  
 /ALL

PAGE: 001

MEMBER NAME	TYPE	VERSION	YR/DAY	HH MM:SS	USER DATA	SUB-DIV	TOP OF SUB-DIV
\$OSYSTAB	ABS	00000	73/027	16:33:55	0290 0F00 0000 0000 0000	01	00000000
						02	00000004
						03	00000000
\$OSYSYB1	ABS	00000	73/027	16:34:00	007C 0F00 0290 0290 0000	01	00000000
						02	00000008
						03	00000000
\$OSYSTB2	ABS	00000	73/027	16:34:04	0150 0F00 030C 030C 0000	01	00000000
						02	00000010
						03	00000000
\$IODRV	ABS	00000	73/027	16:34:13	070C 0000 045C 045C 0000	01	00000000
						02	00000013
						03	00000000
\$TPCDRVD	ABS	00000	73/027	16:34:17	006A 0C00 0C32 0C32 0000	01	00000000
						02	00000023
						03	00000000
\$TPCDRVC	ABS	00000	73/027	16:34:20	00B4 0C00 0C32 0C32 0000	01	00000000
						02	00000025
						03	00000000
\$TPCDRVB	ABS	00000	73/027	16:34:24	00B4 0C00 0C32 0C32 0000	01	00000000
						02	00000027
						03	00000000
\$DMCV09	OBJ	00000	73/002	10:13:27		01	00000959
						02	00000961
						03	00000966
						04	00000000
\$ERDC1C	OBJ	00000	73/002	10:17:06		01	00000968
						02	00000970
						03	00000972
						04	00000000
D \$ERRP53	OBJ	00000	73/002	10:17:06		01	00000974
						02	00000976
						03	00000978
						04	00000000
\$ERRES	OBJ	00000	73/002	10:17:06		01	00001024
						02	00001026
						03	00001028
						04	00001030

2-18

Figure 2-2. Sample PTOC Listing

## COPY LIBRARY MEMBER (COPY)

The COPY function places the active (non-deleted) members of one library on another library. Individual members of the library being copied may be specifically included with or excluded from the COPY function. The receiving library may be either a library already in use, or a newly allocated library, but the block size of both the input and output libraries must be identical. Whenever members of a library are being copied to a library already in use, they are placed on that library following all members of the receiving library. If a member on the receiving library bears the same name and type as that of a copied member, and protection has not been specified, the pre-existing member is marked for deletion, leaving the copied member as current. Deleted members of the library being copied are never included in the COPY function.

A library may be created with the COPY function as a backup for the original library. Members being copied may be renamed by specifying a different output member name for that member when it is copied. In such a case, the candidate for deletion on the receiving library is the member bearing the same name and type as specified by the output member name, subject to the protection specification.

The content of the //PAR statement for the COPY function is:

```
COMMAND=COPY
[,ILIB=input library identifier]
[,OLIB=output library identifier]
[,MEM=(input member name[,type] [,output member name] [,type] [,P] )]
[,MTYPE=member type]
[,SELECT= { I
           E } ]
[,LIST= { YES
         NO } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE= { 1
          2
          3 } ]
[,TITLE='literal string']
```

The COMMAND=COPY keyword-operand is required. All other keywords are optional, with default provided except for MEM and MTYPE.

The default values listed in Table 2-3 should be used whenever possible for the COPY function.

The MEM keyword is used to specify the names of input members to be either included or excluded in the copy function, as specified with the SELECT keyword or its default. When the output member name is specified, it will be used in place of the input member name for that copied member only, on the receiving library. If a member on the output library has the same name and type as a member being copied (as specified by the output member name or, in its absence, the input member name), the existing member will be marked for deletion unless the protection key, P, is specified. If this situation occurs and protection is specified, the program aborts.



**Table 2-3. Default Values for LIBUTIL COPY Function**

Keyword	Default
ILIB	INPUT
OLIB	OUTPUT
SELECT	I when MEM is coded; otherwise E
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

Multiple MEM keywords, one for each specified member, may be used. When type is omitted from the MEM operand for any named member, the MTYPE keyword-operand must be specified. Since MTYPE can be specified only once, all members whose types are omitted from the MEM operand must be the same. To reduce coding and overhead, MTYPE should be used in place of type whenever a number of members are being specified with the same member type. LIST specifies whether or not the names of copied members are to be listed, as they are copied. ID=LIST must appear on a //DEF statement in any library utility run, since the Librarian always attempts to print the function requests and its responses.

The following are examples of //PAR statements that request the LIBUTIL COPY function.

**Example 1:**

In this example all non-deleted members of the library specified with ID=INPUT on a //DEF statement in the step will be copied to the library specified with ID=OUTPUT.

```
//PAR COMMAND=COPY
```

**Example 2:**

In the next example members of the library specified by ID=INPUT, except deleted members and the source members specified to be excluded, are copied to the library specified by ID=OUTPUT.

```
//PAR COMMAND=COPY, MEM=RA1, MEM=FA2,
//PAR MEM=RA18, MEM=RX2, MEM=RQ3,
//PAR SELECT=E, MTYPE=SRC
```

The following examples illustrate the Control Language statements of a step which uses the COPY functions.

### Example 3:

In this example all non-deleted members of library LIB620 will be copied to library LIB621. No list of copied members will be produced, although the Librarian will record parameters received.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=COPY,LIST=NO
//DEF  ID=INPUT,FIL=LIB620,STA=(P,I)
//DEF  ID=OUTPUT,FIL=LIB621,STA=(P,O)
//DEF  ID=LIST,DEV=PRT
//EOJ
```

### Example 4:

In the final example members of library PAYLIB3 will be copied to library PERS27, except source members PAY6 and PAY32 which are excluded. A listing showing names of each copied member will be made.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=COPY,MEM=(PAY6,SRC),
//PAR  MEM=(PAY32,SRC),SELECT=E
//DEF  ID=LIST,DEV=PRINTER
//DEF  ID=INPUT,FIL=PAYLIB3,STA=(P,I)
//DEF  ID=OUTPUT,FIL=PERS27,STA=(P,O)
//EOJ
```

## DELETE LIBRARY MEMBER (DELETE)

The DELETE function flags the library directory entries of named members as deleted. Members marked as deleted are ignored in all LIBUTIL functions except that names of deleted members will appear on the listing displayed by the PTOC function.

The areas of the library and its directories occupied by deleted members remain unavailable, unless a PACK function is performed to remove the deleted members and compress the library or a COPY is executed to build a new library that excludes the deleted members.

The content of the //PAR statement used for the DELETE function is:

```
COMMAND=DELETE
,MEM=(member name[,type] )
[,OLIB=library identifier]
[,MTYPE=member type]
[ ,LIST= { YES }
         { NO } ]
[,INITPG=initial page number]
```

*(continued next page)*

[,PGSIZE=lines per page]

[,SPACE= $\left. \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \right\}$ ]

[,TITLE='literal string']

The default values listed in Table 2-4 should be used whenever possible for the DELETE function.

Table 2-4. Default Values for LIBUTIL DELETE Function

Keyword	Default
OLIB	OUTPUT
LIST	YES (does not include the deleted members in the listing)
INITPG	1
PGSIZE	60
SPACE	1
TITLE	System header line

The MEM keyword is required and uses only two operand fields for the DELETE function. The operands are the name of the member to be marked as deleted, and its type. MTYPE is used only when the type is omitted from MEM, and is then required.

The following are examples of //PAR statements that request the LIBUTIL DELETE function.

Example 1:

In this example, the source member ADMIN7 will be marked deleted from the library specified with ID=OUTPUT on the //DEF statement.

```
//PAR COMMAND=DELETE,  
//PAR MEM=(ADMIN7,SRC)
```

Example 2:

In this example object members COR16, COR23, COR39, and COR57 will be marked deleted from the library specified by ID=MYFILE in the step.

```
//PAR COMMAND=DELETE,MTYPE=OBJ,  
//PAR MEM=COR16,MEM=COR23,MEM=COR39,  
//PAR MEM=COR57,OLIB=MYFILE
```

The following examples show Control Language statements for a job step using the DELETE function.

**Example 3:**

In this example, two cataloged procedures, ADD and SUBT, are marked deleted in library A120. Names of members will be listed as they are deleted. The ID might be specified as PART1 instead of OUTPUT for convenience in a multi-function step.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=DELETE,OLIB=PART1,
//PAR  MEM=ADD,MEM=SUBT,MTYPE=PRO
//DEF  ID=PART1,FIL=A120,STA=(P,O)
//DEF  ID=LIST,DEV=PRINTER
//EOJ
```

**Example 4:**

In this example, source members CHECK4 and CHECK12 will be marked deleted in library CHECKING. There is no listing of members as they are deleted, though the Librarian will list a summary of parameters received.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//DEF  ID=LIST,DEV=PRT
//DEF  ID=OUTPUT,FIL=CHECKING,STA=(P,O)
//PAR  COMMAND=DELETE,LIST=NO,
//PAR  MEM=(CHECK4,SRC),MEM=(CHECK12,SRC)
//EOJ
```

**COMPRESS LIBRARY (PACK)**

The PACK function compresses a library, removing areas previously assigned to members that have been marked for deletion as a result of other LIBUTIL functions. When a member is to be removed from a library, its entry in the library catalog is marked deleted, making the space it occupies inaccessible.

PACK copies all non-deleted members from the specified library to a named intermediate file but does not copy deleted members. The program then reinitializes the library specified by OLIB and copies those members back to the output file, including or excluding (according to the SELECT keyword or its default) members named with MEM keywords. Thus the space formerly occupied by deleted members becomes available for use, and is located at the end of the library. Members copied back under the SELECT=I option assume a sequence on the output library corresponding to the order of their appearance on //PAR cards. The intermediate file used in the PACK may be specified as a permanent file and may be retained as backup. This backup may be critical in case of an I/O error or system crash occurring while the PACK function is in progress, since the input/output file, having been reinitialized at the beginning of the copy back portion of the step, may not be left in a

usable state by such an event. The intermediate file specified by WLIB is initialized during the PACK function; therefore, an existing library with usable information in it should not be used for WLIB.

The content of the //PAR statement used for the PACK function is:

```

COMMAND=PACK
[,OLIB=input/output library identifier]
[,WLIB=intermediate work library]
[,MEM=(input member name[type] [,output member name] [,type] [,P] )]
[,MTYPE=member type]
[,SELECT= { I } ]
           { E } ]
[,LIST= { YES } ]
         { NO } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE= { 1 } ]
          { 2 } ]
          { 3 } ]
[,TITLE='literal string']

```

The default values listed in Table 2-5 should be used whenever possible for the PACK function.

Table 2-5. Default Values for LIEUTIL PACK Function

Keyword	Default
OLIB	OUTPUT
WLIB	WORK
SELECT	I when MEM is coded; otherwise E
LIST	YES (list the names of compressed members on the LIST output file in addition to the usual listing)
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The MEM keyword is used to specify the names of members to be included in or excluded from the output file of the PACK function. It may also be used to rename members on the output library. If the new name and its type match the name and type of a member already copied from the intermediate file during the same PACK operation, the earlier member is marked for deletion. However, if protection was specified, the Librarian aborts the run. Member names on the intermediate library are the input library names. Multiple MEM keywords may be used.

MTYPE is used to specify member type for all member names for which type is omitted. MTYPE should be used in place of type on MEM whenever there are a number of members of one type being specified. Only one MTYPE keyword-operand may be used.

LIST=YES calls for listing the names of copied members as the Librarian copies them onto the intermediate file, and listing the names of the members retained on the compressed output library (copied back from the intermediate library). This listing indicates the progress of the run and the state of the two libraries, should the run be interrupted by an I/O failure or other circumstance.

The following examples show the Control Language statements of a job step which uses the PACK function.

#### Example 1:

In this example all non-deleted members of library BILM6 are copied to library BILM61, and back to BILM6. Deleted entries in the library catalog are removed. Since LIST=NO is specified, names of members copied to BILM61 and back to BILM6 are not listed. This coding is not advised for PACK, for the reasons stated in the preceding paragraph.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=PACK,LIST=NO
//DEF  ID=OUTPUT,FIL=BILM6,STA=(P,O)
//DEF  ID=WORK,FIL=BILM61,STA(P,O)
//DEF  ID=LIST,DEV=PRT
//EOJ
```

#### Example 2:

In this example all non-deleted members of library INV60 are copied to library INV70. The named members only, and their directory entries, are copied from INV70 back to library INV60. The original and final sequence of non-deleted members on INV60 can be seen by a sample Librarian listing for this run. Note the members copied out to INV70 but not copied back to INV60 due to the SELECT default (inclusive). Note also that where an output member name is specified (BALT in the example), that name is shown on the listing as the member copied back.

```
//JOB  NAME=EXAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=PACK
//PAR  MEM=B1,MEM=B2,MEM=(B3,,BALT),
```

```
//PAR MEM=B4,MEM=B7,MEM=B8,
//PAR MEM=B9,MEM=B12,MTYPE=SRC
//DEF ID=LIST,DEV=PRINTER
//DEF ID=OUTPUT,FIL=INV60,STA=(P,O)
//DEF ID=WORK,FIL=INV70,STA=(P,O)
//EOJ
```

Sample Pack Listing

B3	SRC	SAVED ON WLIB
B9	ABS	SAVED ON WLIB
B13	SRC	SAVED ON WLIB
SRC1	SRC	SAVED ON WLIB
B1	SRC	SAVED ON WLIB
B9	SRC	SAVED ON WLIB
B4	SRC	SAVED ON WLIB
B8	SRC	SAVED ON WLIB
B7	SRC	SAVED ON WLIB
B8	SRC	SAVED ON WLIB
B8	OBJ	SAVED ON WLIB
B12	SRC	SAVED ON WLIB
B1	SRC	PACKED INTO OLIB
B2	SRC	PACKED INTO OLIB
BALT	SRC	PACKED INTO OLIB
B4	SRC	PACKED INTO OLIB
B7	SRC	PACKED INTO OLIB
B8	SRC	PACKED INTO OLIB
B9	SRC	PACKED INTO OLIB
B12	SRC	PACKED INTO OLIB

**ASSIGN NEW MEMBER NAME (RENAME)**

The RENAME function assigns a new name to a member of a specified library. The catalog entry of the old member name is marked deleted and a new catalog entry created for the new name. The data for the member in the library remains unchanged and available for use. The space in the catalog occupied by the old member name is unavailable until a PACK or COPY function is performed, replacing the contents of the library directory. Until that time, the old name is still shown in PTOC lists, preceded by the letter D, along with the new one.

The content of the //PAR statement used for the RENAME function is:

```
COMMAND=RENAME
,MEM=(old member name,[type],new member name,[type] [,P])
[,OLIB=library identifier]
[,MTYPE=member type]
[,VERSION=version number]
[,LIST= { YES }
        { NO } ]
```

*(continued next page)*

[,INITPG=initial page number]  
 [,PGSIZE=lines per page]  
 [,SPACE=  $\left. \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \right\}$  ]  
 [,TITLE='literal string']

The default values listed in Table 2-6 should be used whenever possible for the RENAME function.

**Table 2-6. Default Values for LIBUTIL RENAME Function**

Keyword	Default
OLIB	OUTPUT
VERSION	Entry currently in version field of old member entry
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The MEM keyword is required and specifies the old member name (to be deleted) and the new member name. If P is not coded, a new member name and type identical with a non-deleted member name and type on the library will cause the entry already on the library to be marked deleted. If P is coded and the preceding situation occurs, the run aborts without performing the RENAME. VERSION may be used for convenience of identifying a member on a listing, but is not part of the name. If not specified, the version of the old member will be used.

The following are examples of //PAR statements that request the LIBUTIL RENAME function.

**Example 1:**

In this example, source member A of the library specified by ID=OUTPUT on a //DEFINE Control Language statement in the step will be renamed A7. However, if source member A7 already exists on that library, it will be protected, the RENAME will have no effect, and the run will abort. A7 will have the same version number as A.

```
//PAR  COMMAND=RENAME,MEM=(A,SRC,A7,SRC,P)
```



### Example 2:

In this example, member RES of the library specified with ID=OUTPUT is renamed BLK, VERSION 04. An existing source member named BLK is not protected.

```
//PAR  COMMAND=RENAME,  
//PAR  MEM(RES,SRC,BLK,SRC),VERSION=(04
```

The following examples show the Control Language statements of a job step using the RENAME function.

### Example 3:

In this example four macro members of library MACROFIL are renamed. Version numbers on all the new member names will be the same as those on the old members. Existing members of the macro type with the names CHG1, CHG16, CHR7, and CHR11 are not protected.

```
//JOB  NAME=SAMPLE  
//EX   PGM=LIBUTIL  
//PAR  COMMAND=RENAME,MTYPE=MAC,  
//PAR  MEM=(MAC1,,CHG1),MEM=(MAC16,,CHG16),  
//PAR  MEM=(MAR7,,CHR7),MEM=(MAR11,,CHR11)  
//DEF  ID=LIST,DEV=PRINTER  
//DEF  ID=OUTPUT,FIL=MACROFIL,STA=(P O)  
//EOJ
```

### Example 4:

In this example the members SAV and SAV2 of library SPC123 are renamed BRS6 and BRS5 with a version number 27. Member names are not listed in the course of the RENAME execution. The ID of SPC123 is INPUT6 instead of the default OUTPUT. An existing source member named BRS6 would be protected, but one named BRS5 would not.

```
//JOB  NAME=SAMPLE  
//EX   PGM=LIBUTIL  
//PAR  COMMAND=RENAME,OLIB=INPUT6,  
//PAR  MEM=(SAV,SRC,BRS6,SRC,P),  
//PAR  MEM=(SAV2,SRC,BRS5,SRC),  
//PAR  VERSION=27,LIST=NO  
//DEF  ID=INPUT6,FIL=SPC123,STA=(P,O)  
//DEF  ID=LIST,DEV=PRT  
//EOJ
```

## PUNCH ENCODED MEMBER (DUMP)

The DUMP function converts the members of a relocatable or absolute load or object library to a sequential punched card deck. Members may be dumped in either LIBUTIL reloadable or machine loadable format (see Appendix D). A member may be dumped in machine loadable format only if it is an absolute load module, a stand alone program which can be reset loaded. Each member dumped in reloadable format is preceded by a unique member identification card (or card image). Each member dumped in machine loadable format is preceded by a separator card.

The content of the //PAR statement for the DUMP function is:

```
COMMAND=DUMP
,MEM=(input member name[,type])
[,ILIB=input library identifier]
[,OFIL=output dumped file identifier]
[,MTYPE=member type]
[,MODE= { R
          M } ]
[,LIST= { YES
          NO } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE= { 1
            2
            3 } ]
[,TITLE='literal string']
```

The default values listed in Table 2-7 should be used whenever possible with the DUMP function.

Table 2-7. Default Values for LIBUTIL DUMP Function

Keyword	Default
ILIB	INPUT
OFIL	SEQOUT
MODE	R
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The COMMAND=DUMP keyword-operand is required. In addition, the MEM keyword must be used to specify the names of the members of the library to be dumped. The MTYPE keyword is used only when the type operand of one or more MEM keywords is omitted, and then it is required. ILIB specifies the library from which members are to be dumped. OFIL designates the file to which the members are to be dumped. OFIL designates the file to which the members are to be dumped. MODE specifies reloadable or machine loadable dump format. Machine loadable (M) format is valid only for stand-alone programs (those that do not require the operating system) which are stored as absolute load modules and may be reset loaded on the MRX/40 and 50 Systems.

The MODE=R member identification card has the following format:

<u>Column</u>	<u>Contents</u>
1	Hexadecimal DD, dump output identifier
2	Hexadecimal FF, member identification card code
3-6	Text header (common stored data format header)
7-14	Member name
15	Member Type
16	Reserved
17-18	Attribute field
19-20	Version
21	Number of user extension words
22	Number of sub-division links
23-25	Creation date
26-29	Creation time
30-49	User extensions
50-76	Reserved
77-80	Sequence number

Information in the identification card is obtained from the library directory (Appendix A).

The MODE=M member separator card contains zeros in all 80 columns. It is ignored when the deck is loaded, and simply provides a means of separating decks.

The following are examples of //PAR statements that request the LIBUTIL DUMP function.

**Example 1:**

In this example, object member AGR will be dumped in reloadable format from the library specified by ID=INPUT to the file specified by ID=SEQOUT.

```
//PAR  COMMAND=DUMP,MEM=(AGR,OBJ)
```

**Example 2:**

In this example an absolute load module, BR620, is dumped in machine loadable format from the library specified by ID=INPUT to the file specified by ID=SEQOUT. BR620 must be a stand-alone program that can be initiated via reset-load.

```
//PAR COMMAND=DUMP,MEM=(BR620,ABS),
//PAR MODE=M
```

The following examples show the Control Language statements for steps which use the DUMP function of LIBUTIL.

#### Example 3:

In this example five relocatable load members of library BUSAD236 will be dumped in LIBUTIL reloadable format onto a punched card file. Their names will not be listed as they are dumped, but they will appear in the summary list of parameters received.

```
//JOB NAME=SAMPLE
//EX PGM=LIBUTIL
//DEF ID=LIST,DEV=PRT
//DEF ID=SEQOUT,DEV=READPUNCH
//DEF ID=INPUT,FIL=BUSAD236,STA=(P,O)
//PAR LIST=NO,COMMAND=DUMP,
//PAR MEM=AD6,MEM=AD7,MEM=AD8,
//PAR MEM=AD9,MEM=AD10,MTYPE=REL
//EOJ
```

#### Example 4:

In a system with a reader-punch as the input reader, punching must be done via SYSCRD, as in this example. Absolute load member SHOP6 and object member SHOP6 are dumped from library SHOPCHART2 to a punched card file. The names of both members are listed as they are dumped.

```
//JOB NAME=SAMPLE
//EX PGM=LIBUTIL
//PAR COMMAND=DUMP,ILIB=OUTPUT,
//PAR MEM=(SHOP6,ABS),MEM=(SHOP6,OBJ)
//DEF ID=LIST,DEV=PRINTER
//DEF ID=SEQOUT,DEV=SYSCRD
//DEF ID=OUTPUT,FIL=SHOPCHART2,STA=(P,O)
//DATA FIL=SYSCRD
.
.
.
(Blank cards)
.
.
.
//EOJ
```

## LOAD DUMPED MEMBER (LOAD)

The LOAD function restores one or more members dumped with MODE=R to a library. The card deck, magnetic tape, or sequential disc file previously generated by the DUMP function is used as input to recreate the member via the LOAD function. There is no protection of existing members on the library from being deleted by loading members of the same name and type. Name and type of the input members are derived from the member identification cards provided when the members were dumped.

Members can be renamed before the LOAD function by changing the name in the identification card. Members dumped in machine-loadable format cannot be reloaded on a library using the LOAD function.

The content of the //PAR statement for the LOAD function is:

```
COMMAND=LOAD
[,IFIL=input file identifier]
[,OLIB=output file identifier]
[,LIST= { YES
         NO } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE= { 1
           2
           3 } ]
[,TITLE='literal string']
```

The default values listed in Table 2-8 should be used whenever possible with the LOAD function.

Table 2-8. Default Values for LIEUTIL LOAD Function

Keyword	Default
IFIL	SEQIN
OLIB	OUTPUT
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The COMMAND=LOAD keyword-operand is required. IFIL specifies the input file (card, tape, disc) containing the dumped members to be loaded and recreated. OLIB specifies the library onto which those members will be loaded. The recreated members and their directory entries are loaded at the end of the library, immediately following the last member of that library. Any member already on the library is marked for deletion if it bears the same name and type as an incoming member being loaded. The data separator statement (/\*LIB) must not appear between members to be loaded by one LOAD command.

The following are examples of //PAR statements that request the LIBUTIL LOAD function:

**Example 1:**

In this example the members on the file specified with ID=SEQIN will be loaded onto the library specified with ID=OUTPUT.

```
//PAR  COMMAND=LOAD
```

**Example 2:**

In this example the members of the file specified with ID=SEQIN will be loaded onto the library specified with ID=LODLIB.

```
//PAR  COMMAND=LOAD,OLIB=LODLIB
```

The following examples show the Control Language statements of job steps using the LOAD function.

**Example 3:**

In this example members A and B in the card reader file, SEQIN, are loaded onto the library LODLIB27. The header line is specified by the programmer. The listing will be double-spaced and will identify each member by name and type as it is loaded. The data separator statement, /\*LIB, is not required at the end of the file, but is required to separate sets of data for different commands in the same data file.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//DEF  ID=LIST,DEV=PRINTER
//DEF  ID=SEQIN,FIL=RELOAD
//DEF  ID=OUTPUT,FIL=LODLIB27,STA=(P,O)
//PAR  COMMAND=LOAD,SPACE=2,
//PAR  TITLE='LODLIB27 LOAD'
//DATA FIL=RELOAD
```

(Dumped deck of member A)

(Dumped deck of member B)

```
/*LIB
/*
//EOJ
```

#### Example 4:

In this example the members of a tape file, R726LOD3 with volume identifier 1473 are loaded onto disc file ARCHR3. Member names will not be listed as they are loaded, but the Librarian will produce a listing acknowledging the LOAD command and function complete.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=LOAD,LIST=NO
//DEF  ID=SEQIN,FIL=R726LOD3,DEV=TAPE16,VOL=1473
//DEF  ID=OUTPUT,FIL=ARCHR3,STA=(P,O)
//DEF  ID=LIST,DEV=PRT
//EOJ
```

#### MODIFY LOAD MEMBER (PATCH)

The PATCH function is used to modify absolute and relocatable load members of libraries. Each modification processed becomes a permanent change to the member module. That is, the modification is done in place in the library and the original member data is no longer available. The PATCH routine can also be directed to verify the contents of the module prior to modification. (See the Linkage Editor section of this manual, Section 3, for the structure of a load module.)

The contents of the //PAR statement for the PATCH function are:

```
COMMAND=PATCH
,MEM=(input member name[,type] )
[ ,MTYPE= { ABS
           REL } ]
[,ULIB=update library identifier]
[,IFIL=input file identifier]
[ ,LIST= { YES
         NO } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[ ,SPACE= { 1
           2
           3 } ]
[,TITLE='literal string']
```

The default values in Table 2-9 should be used whenever possible for the PATCH function.

**Table 2-9. Default Values for LIBUTIL PATCH Function**

Keyword	Default
ULIB	UPDATE
IFIL	SEQIN
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The **COMMAND=PATCH** keyword-operand is required. The **MEM** keyword must also be included, specifying the member to be patched. The member type must be specified, either with the **MTYPE** keyword or as an operand to the **MEM** keyword. The only valid member types for the **PATCH** command are **ABS** and **REL**. Members of these types are produced by the Linkage Editor.

The **PATCH** directives are presented as data to the **PATCH** routine in the data files specified by the **IFIL** keyword. The general format is:

command displacement text

A single space precedes and follows the displacement field. Multiple text sub-fields are separated by commas, a comma following each sub-field except the last. The text is coded as hexadecimal data and must be specified in words (2 hexadecimal characters per byte, 2 bytes per word).

The displacement must be coded as a hexadecimal value equal to the displacement from the beginning of the load module relative to zero. If the displacement specified is outside of the text for the named member, the utility will be terminated with an error code.

There are two commands, **VER** and **REP**. **VER** directs the **PATCH** routine to verify that the contents of the member beginning at the designated displacement is equal to the specified text. An unequal compare will result in termination of the utility. **REP** directs the **PATCH** routine to replace the contents of the member beginning at the designated displacement with the specified text. Separate formats are provided for relocatable and absolute member patches.

### **Patching Relocatable Load Modules**

Patch words for relocatable members can be specified for Absolute Text Word Attribute (**A**) or Relocatable Text Word Attribute (**R**). If **R** is specified, the relocatable program loader will perform relocation adjustment at load time. The format for relocatable member patches is as follows:



$\left\{ \begin{array}{l} \text{VER} \\ \text{REP} \end{array} \right\}$  displacement text,  $\left\{ \begin{array}{l} \text{A} \\ \text{R} \end{array} \right\}$ ,text,  $\left\{ \begin{array}{l} \text{A} \\ \text{R} \end{array} \right\}$ , . . . ,text,  $\left\{ \begin{array}{l} \text{A} \\ \text{R} \end{array} \right\}$

One word or several consecutive words of text beginning at the same displacement may be specified with each command. A comma follows each text word and each attribute code except the last. The following examples illustrate the format.

```
VER 016E F0F0,A
REP 016E F1F1,A,0645,R
```

### Patching Absolute Load Modules

In the format for absolute members, consecutive patch words from a single displacement are separated by commas. No attribute codes are provided for patches to absolute members. The format for absolute member patches is as follows:

$\left\{ \begin{array}{l} \text{VER} \\ \text{REP} \end{array} \right\}$  displacement text, . . . ,text

This provides for one or more consecutive words of text beginning at one displacement, for example:

```
VER 016E F0F0
REP 016E EC00,0644
```

### PATCH Examples

The following are examples of //PAR statements that request the LIBUTIL PATCH function.

#### Example 1:

In this example absolute member STOR6, located on the library specified by ID=UPDATE will be patched using the directives and data in the data file specified by ID=SEQIN on its //DEFINE statement.

```
//PAR COMMAND=PATCH,MEM=(STOR6,ABS)
```

#### Example 2:

In this example a relocatable member of the library specified with ID=OUTPUT on its //DEFINE statement will be patched. The member name is PARTS. The data file is specified with ID=SEQIN on its //DEFINE statement. A listing will be produced showing the input parameters, but not the patch directives performed.

```
//PAR COMMAND=PATCH,ULIB=OUTPUT,
//PAR MEM=(PARTS,REL),LIST=NO
```

The following examples show the Control Language statements of a step that uses the PATCH function.

Example 1:

In this example absolute member PRO7 of library LOADLIB is patched via the directives in the data file PATCHLOAD. A listing showing input parameters and patch directives will be produced.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//DEF  ID=LIST,DEV=PRT
//DEF  ID=UPDATE,FILE=LOADLIB,STA=(P,O)
//DEF  ID=SEQIN,FIL=PATCHLOAD
//PAR  COMMAND=PATCH,MEM=(PRO7,ABS)
//DATA FIL=PATCHLOAD
```

(Patch directives)

```
/*LIB
/*
//EOJ
```

Example 2:

In this example NUM7, a relocatable member of library LODLIB12, is patched using directives in data file SETUP. A listing showing the input parameters will be produced, but the patch directives performed will not be listed.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  MEM=(NUM7,REL),
//PAR  LIST=NO,
//PAR  COMMAND=PATCH,
//PAR  ULIB=IN3,
//PAR  IFIL=BLDUP
//DEF  ID=IN3,FIL=LODLIB12,STA=(P,O)
//DEF  ID=BLDUP,FIL=SETUP
//DEF  ID=LIST,DEV=PRT
//DATA FIL=SETUP
```

(Patch directives)

```
/*LIB
/*
//EOJ
```

## PRINT SYMBOLIC MEMBER (PRINT)

The PRINT function prints the named members of a symbolic (source, macro, or procedure) library, with data of the member displayed in alphanumeric character representation. Any bit combinations not equivalent to a printable EBCDIC character will be shown as blank on the listing.

The printed output consists of the LIBUTIL header (system date, time, LIBUTIL function, member identification, and page number) or the optional user-specified header, and the data of the member.

The listing produced by the PRINT function will be output by the //DEF Control Language statement specifying ID=LIST. LIST=YES must be used with the PRINT command, either specified on a //PAR statement or by default.

The content of the //PAR statement of the PRINT function of LIBUTIL is:

```
COMMAND=PRINT
,MEM=(member name [type])
[,ILIB=library identifier]
[,MTYPE=member type]
[,LIST=YES]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE= { 1
           2 } ]
[,TITLE='literal string']
```

The default values listed in Table 2-10 should be used whenever possible for the PRINT function.

Table 2-10. Default Values for LIBUTIL PRINT Function

Keyword	Default
ILIB	INPUT
LIST	YES (LIST=NO is illegal for PRINT)
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The COMMAND=PRINT keyword-operand is required. The MEM keyword must also be used for each member to be printed. Multiple MEM keywords are used to print more than one member. The MTYPE keyword is used only when type is omitted from MEM, and then is required.

The following examples show the Control Language statements of steps that use the PRINT function.

#### Example 1:

In this example six members, GR631–GR636, of library GRAIN76 will be printed double-spaced. There will be 50 lines to the page. The LIBUTIL header will be used.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//DEF  ID=LIST,DEV=PRINTER
//DEF  ID=INPUT,FIL=GRAIN76,STA=(P,I)
//PAR  COMMAND=PRINT,PGSIZE=50,SPACE=2,
//PAR  MEM=GR631,MEM=GR632,
//PAR  MEM=GR633,MEM=GR634,
//PAR  MEM=GR635,MEM=GR636,MTYPE=SRC
//EOJ
```

#### Example 2:

In this example four procedure members of library EXCHANGE61 will be printed single-spaced. There will be 60 lines to a page. Pages will be numbered from 600. The header to be printed is specified by the programmer.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=PRINT,MTYPE=PRO,
//PAR  TITLE='EXCHANGE INFORMATION FILE 73',
//PAR  MEM=STOCK1,MEM=SECUR6,INITPG=600,
//PAR  MEM=BOND29,MEM=YIELD17
//DEF  ID=LIST,DEV=PRINTER
//DEF  ID=INPUT,FIL=EXCHANGE61,STA=(P,I)
//EOJ
```

#### **PUNCH SYMBOLIC MEMBER (PUNCH)**

The PUNCH function produces a punched card deck or a tape consisting of the card images of a symbolic type (source, macro, or cataloged procedure) member in a library. The output card deck may be resequenced.

The content of the //PAR statement for the PUNCH function is:

```

COMMAND=PUNCH
,MEM=(member name[,type] )
[,OFIL=punch file identifier]
[,MTYPE=type]
[,SEQPOS=(start,length)]
[,NEWSEQ= { (initial number, increment) } ]
[,LIST= { YES } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE= { 1 } ]
[,TITLE='literal string']

```

The default values listed in Table 2-11 should be used whenever possible for the PUNCH function.

The COMMAND=PUNCH keyword-operand is required. The MEM keyword must be used for each member to be punched. MTYPE is used only when type is omitted from MEM and then is required. MTYPE or type with MEM may specify SRC, MAC, or PRO only. The remaining member types (OBJ, ABS, and REL) are illegal for PUNCH.

For resequencing, the start and length of the sequencing field is specified with SEQPOS. The sequence field chosen can be anywhere in the record and can be from 1 to 8 bytes. Neither specification need coincide with the sequence field with which the member was created. The default is column 73, length 8 positions. NEWSEQ specifies the new sequencing values by initial number and increment. If unspecified, renumbering will not occur.

Table 2-11. Default Values for LIEUTIL PUNCH Function

Keyword	Default
ILIB	INPUT
OFIL	SEQOUT
SEQPOS	(73,8)
NEWSEQ	NO
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

The following examples show the Control Language statements of steps that use the PUNCH function.

**Example 1:**

In this example four cataloged procedures, AUTO, AUTO7, AUTO18, and AUTO26 from library AUTOFIL20 are punched on a reader-punch. Renumbering does not occur. Member names are not listed as they are punched.

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//PAR  COMMAND=PUNCH,MEM=AUTO7,
//PAR  MEM=AUTO18,MEM=AUTO26,
//PAR  MEM=AUTO,MTYPE=PRO,LIST=NO
//DEF  ID=LIST,DEV=PRINTER
//DEF  ID=SEQOUT,DEV=READPUNCH
//DEF  ID=INPUT,FIL=AUTOFIL20,STA=(P,I)
//EOJ
```

**Example 2:**

In this example source member ORD6 is transferred from library ORDERLOG to a card image file on magnetic tape specified by ID=PUNCH2. Renumbering is in columns 75-80 beginning with number 1 and incrementing by 10.

```
//JOB  NAME=EXAMPLE
//EX   PGM=LIBUTIL
//DEF  ID=INPUT,FIL=ORDERLOG,STA=(P,I)
//DEF  ID=PUNCH2,DEV=TAPE8
//DEF  ID=LIST,DEV=PRT
//PAR  COMMAND=PUNCH,
//PAR  OFIL=PUNCH2,MEM=(ORD6,SRC),
//PAR  SEQOUT=(75,6),NEWSEQ=(1,10)
//EOJ
```

**CREATE OR MODIFY SYMBOLIC MEMBER (UPDATE)**

The UPDATE function is used to create new symbolic (source, macro, and procedure) members in a library and to modify symbolic members from an existing library. Modification may consist of adding symbolic statements to a member, deleting statements from a member, or combining parts of two or more members within a library. The UPDATE function may use distinct libraries or the same library when modifying a member, producing as output a new member in the output library. Separate //DEFINE cards for ID=ILIB and ID=OLIB are still required even though the same filename is used for both. When the update output library is the same as the input library, the update is not made in place, but it marks the input member for deletion and creates a new member at the high end of the library.

The content of the //PAR statement for the UPDATE command is as follows:

```
COMMAND=UPDATE
,MEM=([input member name],[type],output member name,[type] [,P])
,UMODE= { SEQ
         REL }
[ ,MTYPE= { SRC
          PRO }
         MAC ]
[,ILIB=input library identifier]
[,IFIL=input file identifier]
[,OLIB=output library identifier]
[,SEQPOS=(start,length)]
[ ,NEWSEQ= { (initial number,increment)
            NO } ]
[ ,SEQCHK= { YES
            NO } ]
[,VERSION=version number]
[ ,LIST= { YES
          NO } ]
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[ ,SPACE= { 1
           2
           3 } ]
[,TITLE='literal string']
```

The default values listed in Table 2-12 should be used whenever possible for the UPDATE function.

The COMMAND=UPDATE keyword-operand is required. The UMODE keyword-operand specifies the update method to be used. UMODE=SEQ designates that sequence numbers on the source statements are used in the update, while UMODE=REL specifies that the relative record numbers given on the previous UPDATE or PRINT listing of the member are used. The relative record numbers on the UPDATE listing are not the same as the line numbers on an assembly listing.

The MEM keyword must be specified for the UPDATE function, and must always include an output member name. When the input member name is omitted, creation of a new member (from IFIL) occurs, subject to protection if specified. Otherwise, the input member name specifies an input library member to be processed. Only SRC, PRO, or MAC are legal for the type operand of MEM or MTYPE.

The UPDATE modification process is governed by directive statements. These directives, which for convenience are called pointer directives and copy directives, allow the user to delete from, copy, and insert information into library members.

**Table 2-12. Default Values for LIBUTIL UPDATE Function**

Keyword	Default
UMODE	REL
ILIB	INPUT
IFIL	SEQIN
OLIB	OUTPUT
SEQPOS	(73,8)
NEWSEQ	NO
SEQCHK	NO
VERSION	Entry currently in version field of old member entry
LIST	YES
INITPG	1
PGSIZE	60
SPACE	1 (single space)
TITLE	System header line

For ease of reference, the following narrative in some instances describes the update process as a series of actions on the input library member, although in fact it is not itself modified. All actions on the input library member are a copy from or a failure to copy from ("delete").

### Pointer Directives

The pointer directive, identified by a minus sign in column one followed by one blank, directs the UPDATE program to copy or delete statements from the named member on the primary input library, and to move an internal record pointer for the input library member. One value specified on the directive instructs the program to copy the input member through the specified record; two values separated by a comma instructs the program to delete the records in the inclusive range of the values. The internal record pointer is moved to the record following the last value on the directive.

### Pointer by Relative Record Number

When relative record number mode is selected, either by default or by UMODE=REL, the UPDATE program copies and deletes according to the relative position of the record in the input member. Any data following the directive in the input data stream is then added to the output library member until another directive is encountered. When no additional directives are present in the input data stream and the record pointer is not at the end of the input member, the program copies the remaining records from the input member.



For example, if a user wants to copy records one through four of an existing member, insert two lines of code, and copy the remaining records from the existing member, he must specify a directive for the first four records and include the input data. He need not specify a directive for the balance of the existing member, since the pointer rests at input member record five and the program copies the remaining records. The input would appear as shown below:

```
//DATA FIL=X
- 4          (Copy records 1-4)
Data item 1  { Add code to }
Data item 2  { output member }
/*LIB       (Copy records 5 through end of input library member)
.
.
.
```

If two values separated by a comma are specified, the UPDATE program ignores the records included in the range of values and sets the record pointer to the record following the last value. In effect, these records are deleted from the member on the output library. As in the previous example, assume that the user wants to copy the first four records. However, instead of simply adding the two new lines of code, he wants to replace existing records five and six with the new code and copy the remaining records from the input member. He could accomplish this with the following input:

```
//DATA FIL=X
- 5,6       (Copy records 1-4, delete records 5-6)
Data item 1  { Add code to }
Data item 2  { output members }
/*LIB       (Copy records 7 through end of input library member)
.
.
.
```

In another situation, given the same input member, the user may wish to copy records one through four, insert two lines of code, copy records five through 12, delete records 13 through 15, copy records 16 through 25, replace record 26 with one line of code, and copy the remaining input member records. The directives and data could appear as follows:

```
//DATA FIL=X
- 4          (Copy records 1-4)
Data item 1  { Add code to }
Data item 2  { output member }
- 13,15     (Copy records 5-12, delete records 13-15)
- 26,26     (Copy records 16-25, delete record 26)
Data item 3  (Add line of code)
/*LIB       (Copy remaining records)
.
.
.
```

## 3. LINKAGE EDITOR

### FUNCTIONAL DESCRIPTION

Linkage Editor input may consist of a combination of object modules, load modules, and directives. The primary function of the Linkage Editor is to combine these modules into one or more output load modules, in accordance with the requirements stated on directives. Although this linking or combining of modules is its primary function, the Linkage Editor also:

- Edits modules by replacing, deleting, and rearranging control sections as specified by directives.
- Accepts additional input modules from data sets other than the Primary Input Module, either automatically, or upon request.
- Reserves storage for the COMMON control sections generated by the assembler and the FORTRAN compiler.
- Creates overlay programs (multiple load modules) in a structure defined by directives.
- Provides special processing and diagnostic output options.
- Assigns module attributes that describe the structure, content, and logical format of the output load module.

### MODULE LINKAGE AND EDITING

Linkage Editor processing allows the programmer to divide his program into several modules, each containing one or more control sections. The modules can be separately assembled or compiled. The Linkage Editor combines these modules into one or more load modules with contiguous storage addresses, and resolves all references between modules in the input. The output modules are always placed in a library. The editing functions of the Linkage Editor facilitate program modification. When the functions of a program are changed, the programmer can modify and compile only the affected control sections instead of the entire source module. He can replace, delete, or move control sections through use of the SEG directive.

### ADDITIONAL INPUT SOURCES

Standard subroutines can be included in the output module, thus reducing the work in coding programs. The programmer can specify that a subroutine be included at a particular time during the processing of his program by using a SEG directive. When the Linkage Editor processes a module or a directive file which contains this statement, the module containing the subroutine is retrieved from the indicated input source, and made a part of the output module.

Symbols that are still undefined after all input modules have been processed cause the automatic library search mechanism to search for entry points that will resolve these references. When a module name is found containing the entry point which matches the unresolved symbol, the Linkage Editor processes the module and makes it part of the output program.

## **STORAGE RESERVATION**

The Linkage Editor processes common control sections generated by FORTRAN and the Assembler. The common areas are collected by the Linkage Editor, and a reserved main storage area is provided within the output modules.

## **OVERLAY PROGRAM CREATION**

To minimize main storage requirements, the programmer can organize his program into an overlay structure by dividing it into segments according to the functional relationships of the control sections. Two or more segments that need not be in main storage at the same time can be assigned the same relative storage addresses, and can be loaded at different times.

The programmer uses SEG directives to specify the relationship of segments within the overlay structure. The segments of the program are placed in a library so that loader requests can load them separately when the program is executed. Each load module is placed in the library under a unique member name.

## **SPECIAL PROCESSING AND ERROR DIAGNOSIS**

The programmer can specify special processing options that negate automatic library call or the effect of minor errors. In addition, the Linkage Editor can produce a module map or cross-reference table that shows the arrangement of control sections in the output module and indicates how they communicate with one another. A list of the directives processed can also be produced.

Throughout processing, errors and possible error conditions are printed on the output listing. Fatal errors cause the Linkage Editor to terminate and produce no output module. Additional diagnostic data is automatically logged by the Linkage Editor. The data indicates the disposition of the load module in the output module library.

## **LOAD MODULE ATTRIBUTE ASSIGNMENT**

When the Linkage Editor generates a load module, it places an entry for the module in the directory of the user-defined library. This entry contains attributes that describe the structure, content, and logical format of the load module. The control program uses these attributes to determine what a module contains and how it is to be loaded. Some module attributes can be specified by the programmer; others are specified by the Linkage Editor as a result of information gathered during processing.

## INPUT STRUCTURE

The Linkage Editor receives its input in the form of object modules produced by language processors, primary relocatable load modules produced by previous executions of the Linkage Editor, and directive\* sets in card-image format. The input can be divided into two classifications, basic and secondary.

### BASIC INPUT

Basic input consists of either a Linkage Editor directive set or the primary object module. When there is no directive set, the basic input is a primary object module. The //DEF card\*\* with ID=INPUT names the library file that contains the primary object module, and the operand for the PGM keyword of the //PAR card specifies the cataloged member name of the primary object module.

When the basic input is a directive set, a //DEF card with ID=DIR names a sequential data file on disc storage that contains the directive set. The data file must be in common stored data format, either spooled input or a file created by a utility program. The PGM parameter of the //PAR card specifies the name of the directive set to be used; it must match a name supplied on a NAME directive. The primary input module is identified by the first module name encountered in the highest level SEG directive in the basic input directive set. As in the previous situation, the //DEF card with ID=INPUT names the library file that contains the primary input module. The primary input module can be either an object module or a load module.

### SECONDARY INPUT

Secondary input consists of all object and/or primary relocatable load modules required to become part of the program being link-edited. A primary relocatable load module is one which has a Composite Entry Point List associated with it on a library. It is specified either by external references from the primary object or secondary input modules, or by operand specification of a SEG directive.

An external reference is always made to the symbolic name of an entry point which must be included in the Entry Point List of some object or load module within the Library Search Domain. When the referenced entry point is located, the module in which it is defined is collected into the program being formed. The USE directive assists in the resolution of duplicate entry points.

A SEG directive term may specify either an entire object or load module, or may reference a single control section (CSECT) within a module. The library containing the module must always be included in the current Library Search Domain.

---

\*Directives are discussed in detail later in this section, under the heading *Linkage Editor Directives*.

\*\*Control Language requirements are discussed in detail later in this section, under the heading *Control Language Statement Descriptions*.

## **LIBRARY SEARCH DOMAIN**

In order to locate a required module, the Linkage Editor searches a set of libraries called the Library Search Domain. The specification of this domain may be accomplished in several ways, depending on the LSD parameter of the //PAR card. The library specified by ID=INPUT must contain the Primary Input Module and is always searched first, regardless of the LSD parameter.

The remainder of the domain is searched according to the following conditions:

- If the LSD=NO option is specified, all modules intended to be included in the program must reside on the same library as the primary input module.
- If one or two libraries are specified by LSD=(libname1,libname2), these libraries are searched in the order specified.
- If the LSD parameter is omitted or if AUTO is specified, the system library (\$SYSOBJLIB) containing required system subroutines is searched. Note that if \$SYSOBJLIB is to be included in a specified library search domain with another library other than that specified by ID=INPUT, it must be coded as an operand to the LSD keyword.

Whenever a required module, explicitly defined as a SEG term, is not found within the current Library Search Domain, as defined above, an error message will be displayed and no output module will be produced. In the event that duplicate modules or entry points exist within the current domain, the Linkage Editor will always use the first located in the search hierarchy of modules and libraries specified by the Entry Point Search Domain (described in the following paragraph) and the library search domain. Such duplicates are noted on the link-edit map, but are not treated as errors.

## **ENTRY POINT SEARCH DOMAIN**

The list of load modules to be searched by the Linkage Editor in resolving the external references of a designated load module is called the Entry Point Search Domain (EPSD).

An EPSD should be specified, via the USE directive, whenever externals could be satisfied by more than one entry point within the link-edit map in which a module is to be collected. That is, whenever duplicate entry points exist within a structure and one of them is referenced in a given load module, that module should have an EPSD specified for it. Otherwise the Linkage Editor uses the first satisfactory entry point that it encounters in its search and indeterminate results may occur. The order in which USE directives are entered for a given module specifies the search sequence within the domain.

## EXPRESSIONS

The operand of the SEG directive is in the form of a logical expression composed of a single term or a combination of terms and operators. Spaces may occur any place after the beginning of the operand expression. The operand may not extend into the sequence field of the card (character positions 73-80). Continuation is specified by coding a semicolon into any column preceding column 73 on a card. Scanning continues with the next card, which should not contain a label or a directive. The MRX Assembler does not allow continuation on SEG statements presented to it embedded in assembly language code. The operand expression specifies the desired memory occupation by use of three operators: the plus, the comma, and the parentheses. These designate inclusion, exclusion, and level of occupation, respectively. The operators have the following meanings in SEG directive expressions:

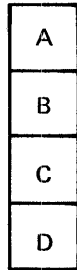
- |                 |   |
|-----------------|---|
| + plus          | Inclusion Operator: Terms separated by a plus sign are considered to be included sequentially in memory in the order encountered. This results in simultaneous memory occupation of the modules named by these terms.   |
| , comma         | Exclusion Operator: Terms separated by a comma are considered to occupy the same memory area. This results in exclusive overlays which are separate load modules.   |
| ( ) parentheses | Grouping or Load Module Operator: The operators for SEG directives have a priority analogous to mathematical symbols. That is, commas are evaluated before pluses, as multiplication is done before addition. Enclosing an expression in parentheses, however, causes evaluation of the enclosed expression prior to evaluating the remaining expression outside the parentheses. In three instances, enclosing expressions within parentheses produces a separate load module: a simple expression (single term) enclosed, a complex expression enclosed in double parentheses, and an enclosed complex expression with a comma preceding the left parenthesis. A complex expression enclosed in parentheses and preceded by a plus does not cause creation of a separate load module, but does cause grouping of the terms within the parentheses prior to evaluation of the remaining expressions. |

The creation of load modules can be illustrated with a few examples. In the examples, diagrams show the level of overlays; that is, which load modules are overlaid by other modules. The space occupied by a module is dependent, of course, on the length of the module. Assume four modules, A, B, C, and D. The following examples show the level of occupation of memory, depending on the arrangement of the operators in the SEG statement.

Example 1:

This statement produces a single load module named ALPHA, composed of the modules A, B, C, and D, as illustrated in the diagram.

ALPHA SEG A+B+C+D

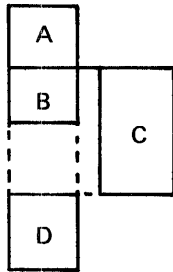


Modules A, B, C, and D will all be loaded together when the load module ALPHA is called. None of the modules can be loaded separately, since only one load module is produced.

Example 2:

This statement produces a root load module named BETA, comprising modules A, B, and D, and an overlay module C, as shown in the diagram.

BETA SEG A+B,C+D

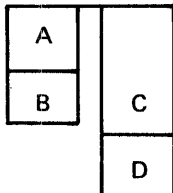


Load module C is link-edited to overlay module B. Module D is link-edited so that it will be loaded with modules A and B, but will occupy space following the area in which load module C will be loaded.

Example 3:

In this statement, modules C+D are enclosed with parentheses and preceded by a comma. Therefore, a load module, named C, will be produced for this expression, as well as a load module named GAMMA, consisting of modules A and B. Load module C will overlay Module B, as shown in the diagram.

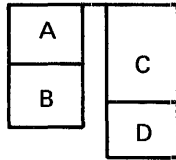
GAMMA SEG A+B,(C+D)



#### Example 4:

In this statement, two load modules are produced, as in the previous statement, the first named DELTA and the second named C. However, the level of storage occupation differs from the previous example in that the load module C will overlay the area occupied by modules A and B of load module DELTA, as illustrated in the diagram.

DELTA SEG (A+B),(C+D)



#### SEG TERMS

Each term of a SEG operand expression consists of from one to three names. These are a control section name, a module name, and a library name. Either a control section name or a module name, or both, must be included as a term of every SEG operand expression. When more than one of these names appears in a single expression, the names are separated by the / character on the SEG card.

#### Control Section Name

A control section name consists of the name of the actual control section or common block as submitted to or generated by a language processor or translator. A control section name must be specified whenever a module name is not included. If specified, the control section name always occurs first in the expression. When both are used, the control section name is followed by a slash and then the module name. A control section name is a 1- to 8-character alphanumeric string. The first character must be alphabetic. This name must be prefixed by the # character to identify it.

#### Module Name

A module name may define the name of an object or load module to be found in the currently specified Library Search Domain or the name of a SEG directive previously encountered in the same object module or directives set in which the reference is made. A module name must be specified whenever a control section name is not included. When both are used, the control section name occurs first, followed by a slash and then the module name. This term is a 1- to 8-character alphanumeric field. The first character must be alphabetic.



## Library Name

A library name specifies the library in which the control section and/or module named in the expression must be located. This term is always optional. Library name defines an exception or override to the normal Library Search Domain. However, the library name specified must be included in the Library Search Domain for the program being generated. When used, the library name follows the module name in the expression. The library name is preceded by a slash. Library name is a 1- to 17-character alphanumeric field with no embedded special characters except dash. The first character may be \$ for system files and libraries only. The library name must be the name of a library specified as the operand of the LSD keyword on the //PAR card.

Terms on a SEG directive may occur in any of the following forms:

#csname — The csname entry specifies a control section or common block name defined in the current module making the reference.

modname — Modname specifies a module name implying all control sections or common blocks contained or defined within it.

#csname/modname — This form designates a specific control section or common block of the named module. In this form modname must be an object module, since the Relocation Dictionary is required to locate the named control section. Modname cannot be a load module or SEG directive.

modname/libraryname — This configuration specifies a module that must be located in a specific library of the Library Search Domain. The normal hierarchy of search is overridden, and only the specified library is searched.

#csname/modname/libraryname — This combination designates that a specific control section or common block of the named object module must be located in the named library. Only this library, which must be included in the Library Search Domain, will be searched. The normal search hierarchy is ignored.

## COMMON ALLOCATION

All common control sections of the same name (whether labeled or 'blank') declared via the Assembler COM instruction, are mapped into the same allocated storage area. Space for a common section will be allocated whenever the first declaration of that common occurs, except in the case of 'blank' common which is always allocated at the end of the module (high order addresses).

Duplicate common definitions with different sizes may exist in independently compiled or assembled programs. However, at link-edit time, only one storage area, with the maximum declared size, is allocated. This is true even though multiple allocations of the common block have been specified in SEG directives.

A labeled common control section may be preset by declaring a CSECT of the same name. Each declaration of that CSECT name presets the area. Therefore, it is extremely important when more than one CSECT is used to preset the common area, that the programmer use caution in specifying linkages to obtain the desired results. In addition, it is recommended that the common area be specified in a resident area that will not be overlaid by other load modules. A blank common is created by use of the COM statement with no label; it has no relationship to a blank CSECT. Blank common cannot be preset; that is, a blank control section declared directly or indirectly cannot be used to preset common.

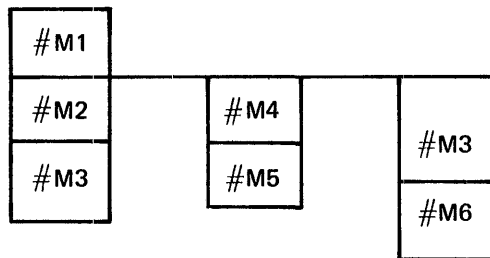
Each CSECT (not declared common) specified in a SEG directive causes storage to be allocated, even though the same CSECT name may be specified more than once in the SEG operand.

**Example:**

In this example control section, #M3 (not declared common) is allocated in two overlay areas, #M2+#M3 and #M3+#M6.

A SEG #M1+(#M2+#M3),(#M4+#M5),(#M3+#M6)

The memory occupation can be illustrated by the following diagram.



**SAMPLE SEG STATEMENTS**

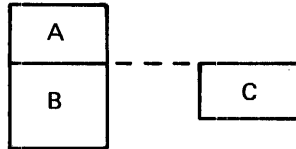
The following are examples of SEG statements used to obtain the memory configurations diagrammed.

In the diagrams, the topmost level indicates the root or main module. Boxes in the same vertical plane as the root module indicate segments that are loaded with the root module. They are not separate modules and therefore cannot be loaded separately. Modules appearing to either side of the root module represent overlays. They are loaded at the relative location calculated by the Linkage Editor. The portions of the root module, or other modules, that they overlay is dependent upon their length. The diagrams use dotted lines to show the locations at which they are loaded relative to the root module.

### Example 1:

SEG Statement:        [label]        SEG        A+B,C

In this example, module A is the root or main segment. Module C overlays B in one memory area. Module B is loaded with root A. Module C is a separate overlay load module, designated by the comma preceding it.



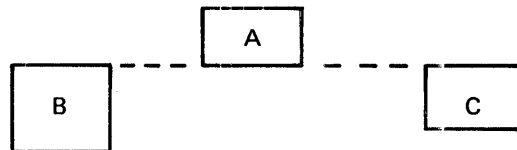
Alternate SEG Statement:        [label]        SEG        A+(B,C)

In this sample statement, the parentheses act as a logical grouping operator and are redundant. The preceding description applies.

### Example 2:

SEG Statement:        [label]        SEG        A+(B),C

In this example, module A is the desired root or main segment. Modules B and C are to overlay one another in the same memory area. Neither B nor C is loaded simultaneously with the root A, because of the parentheses surrounding B and the comma preceding C.



Alternate SEG Statements:        X        SEG        B,C  
   [label]        SEG        A+(X)

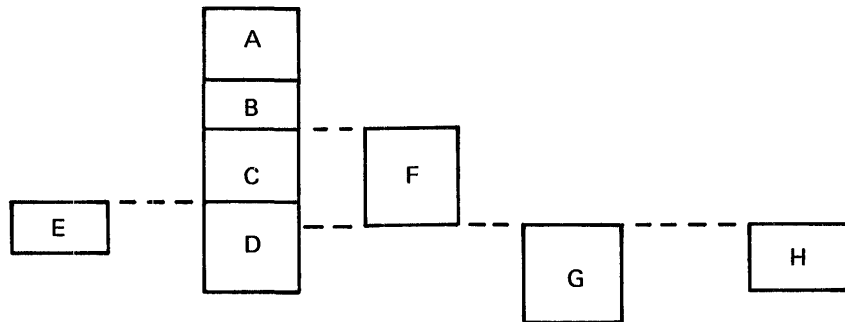
In this alternate example, SEG X defines modules B and C as separate load modules overlaying the same area in memory. Neither B nor C is loaded simultaneously with the root A. The load module shown as B in the illustration will be named X on the library.

Note: Forward SEG references to the label fields of other SEG statements are not allowed. Therefore SEG X must occur prior to the SEG that references it.

### Example 3:

SEG Statement:        [label]        SEG        A+B+(C+D,E),(F+(G),H)

In this sample, modules C and D are loaded with the root, A and B. Module E overlays D. Sub-complex F (modules F, G and H) overlay C and D. Modules E, F, G, and H are all separate load modules.



Alternate SEG Statements:    X        SEG        C+D,E  
                                   Y        SEG        (F)+(G),H  
                                   [label] SEG        A+B+X,Y

In this alternate example, SEG X defines sub-complex C containing modules C, D and E with module E as a separate load module. Modules C and D, as specified, will load with root A and B and are not available as separate load modules.

SEG Y defines sub-complex F containing three separate load modules, G, G and H.

Note: Forward SEG references to the label field of following SEG statements are not allowed. SEG statements X and Y, therefore, must occur physically before the SEG statement in which they are referenced.

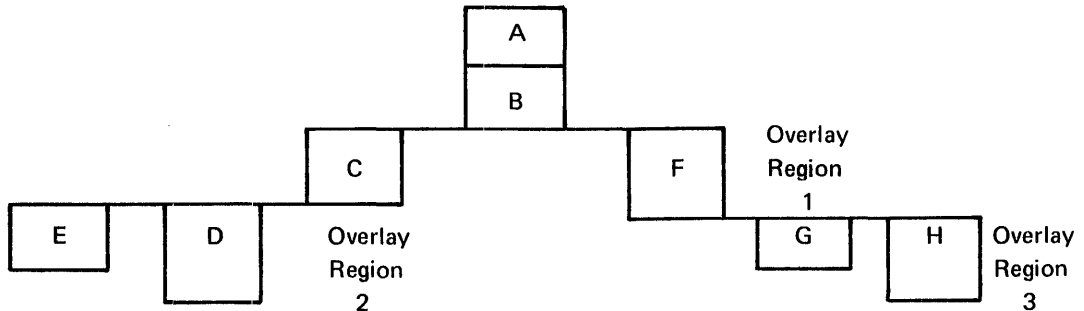
Example 4:

Overlay Region 1

SEG Statement:    [label]    SEG    A+B+((C)+(D),E),(F+(G),H)

Overlay    Overlay  
Region    Region  
2        3

In this example, modules A and B are the root or main segment with two overlays, sub-complexes C and F. Sub-complex C includes module C plus overlays D and E, and sub-complex F includes module F plus overlays G and H.



Alternate SEG Statements:	V	SEG	C+(D),E
	W	SEG	F+(G),H
	[label]	SEG	A+B+(V),W

In this alternate example, SEG V defines sub-complex C containing overlays D and E in addition to module C.

SEG W defines sub-complex F containing overlays G and H in addition to module F.

Parentheses around V, a simple expression, in the last SEG statement causes the sub-complex defined by V to be treated as a separate load module. C is, therefore, a separate load module, as are D (in parentheses) and E (preceded by a comma).

Since the sub-complex defined by W is preceded by a comma in the last SEG statement, F is a separate load module. G and H are separate load modules by virtue of their specification in SEG W.

The last SEG statement specifies the final structure with V and W supplying the sub-complex definition provided on the named statements.

Modules shown as C and F in the diagram will be named V and W respectively on the library, because of their position in their respective SEG statements. Root module A will be named by the label on the last SEG statement. Overlays D, E, G, and H will be cataloged in the library under their given names.

Note: SEG statements V and W must physically precede the statement in which they are referenced.

Alternate SEG Statements:	V	SEG	C+(D),E
	W	SEG	F+(G),H
	X	SEG	V,W
	[label]	SEG	A+B+X

Parentheses around either V in SEG X or C in SEG V or X in the last statement could be used to designate module C as a separate load module.

SEG V defines sub-complex C including load modules C, D, and E with D and E as overlays.

SEG W defines sub-complex F including load modules F, G, and H with overlays G and H.

SEG X defines the relationship between V and W occupying the same memory areas as overlay sub-complexes.

The last SEG statement specifies the final structure with X supplying the sub-complex definitions provided on the named statements.

Module F in the diagram will be named W on the library. Naming of module C depends on the option chosen above.

Since a comma precedes W in SEG X, parentheses around F would be redundant.

SEG statements V and W must precede SEG X which references them. SEG X must occur physically before the last SEG statement.

Alternate SEG Statements:	D	SEG	D,E
	G	SEG	G,H
	X	SEG	(C)+(D)
	F	SEG	F+(G)
	Z	SEG	X,F
	[label]	SEG	A+B+Z

In this example, SEG D specifies the relationship between D and E as overlays. E is defined as a load module.

SEG G specifies the relationship between G and H as overlays. H is defined as a load module.

SEG X defines sub-complex C as containing load modules C and the contents of SEG D as a load module.

SEG F defines sub-complex F as containing module F and the contents of SEG G as a load module.

SEG Z specifies the relationship of SEG X to SEG F as overlays. The content of SEG F is defined as a load module.

The last SEG statement specifies the final structure with Z bringing the sub-complex definitions provided on the named SEG statements.

All modules on the library will be named as shown in the diagram.

## LINK-EDIT MAP

The Linkage Editor creates a listing that includes a heading line, a list of the Linkage Editor Directives included in the input, and a list of the load modules produced, including the name of each load module, its relative relocatable load address, its byte size, and the relative address of its entry point. Under each load module is listed the other control sections, object modules, and other load modules included in the named load module, together with common block names, control section names, and entry points, and their associated addresses. Externals in each module are also listed, showing the external name, the name of the load module containing the entry point that satisfies the external, and the relative address of the entry point.

## TITLE LINE

The title line of the map appears as follows:

```
**LINKAGE EDITOR LEVEL-x      mmddy      hhmss

x          Level designation of the Linkage Editor in use at the site.

mmddy     Current system date (month, day, year).

hhmss    System time (hours, minutes, seconds).
```

## DIRECTIVE LIST

The directive list includes all directives supplied to the Linkage Editor, whether included in a directive set or embedded in the object module. It is essentially a list of the directive card images.

## LOAD MODULE LIST

The load modules are listed as follows:

```
LOAD MODULE= xxxxxxxx BSADR= nnnn SIZE= bbbb ENTRY POINT= aaaa
      zzzzzzzz      nnnn
      CM name      addr
      CS name      addr
      EP name      addr
      PE name      addr
      EX name      addr      modname

xxxxxxx  Name of the load module, as specified by the PGM or XQT
         parameter or by SEG statements.

nnnn     Relocatable load address, relative to a zero base.

bbbb     Composite length of the load module in bytes (described under
         the heading Executable Program Length).

aaaa     Relative relocatable address of the primary entry point of the
         load module (the first entry point if no primary entry point is
         specified). Compilers generate primary entry points according
         to their own rules.

zzzzzzz  Name of an input object module which has been included in
         the load module, and in which the items following the name
         are found.
```

The content of the FETCH macro is as follows:

```
[label]    FETCH    { MOD= symbolic location  
                   of module name  
                   or  
                   ENTRY= symbolic location  
                           of entry point  
                   [ ,ERRCOMP= { YES }  
                           { NO } ]  
                   [ ,LIST= { YES }  
                           { NO } ] }
```

MOD= specifies the address (symbolic location) of the 8-byte field which contains the EBCDIC name of the module to be brought into main storage. This keyword is required for FETCH by module name only, and excludes the use of the ENTRY keyword. The module named at the specified address must reside on the library named as the operand of the LIB keyword on the //EXECUTE statement when the program is executed or on the system load library, \$SYSLODLIB.

ENTRY= specifies the address (symbolic location) of the 8-byte field in main storage which contains the EBCDIC name of the entry point requested. The module containing that entry point will be located and loaded into the program partition of main storage. That module must have been link-edited as one of the segments or overlays of the program currently in execution. This keyword is required for FETCH by entry point, and excludes the use of the MOD keyword.

ERRCOMP=YES specifies that control is to be returned to the requesting program if the service request macro completes with errors. ERRCOMP=NO specifies that control is to be retained by the system in the event of an error, and results in program abort. This keyword is optional; the default is NO. Error completion codes are shown in the macro expansion, Appendix F.

LIST= controls generation of the Service Request and of the parameter string for the macro. YES generates an object string for the macro, but no SR instruction. General register 6 must contain the address of the parameter string when the program is executed. NO generates an SR instruction with no parameter string. Omission of the LIST keyword generates an SR instruction with the macro expansion (parameter string) in line, immediately following the SR.

#### NOTE

Unlike most service request macros, the RETURN keyword is not valid with the FETCH macro. Use of RETURN=YES produces an execution error.



## **SAMPLE FETCH MACRO**

The following is an example of a FETCH macro.

```
LEAP6    FETCH    ENTRY=BAL13,ERRCOMP=YES
```

This example will result in a search of the Composite Entry Point List for the module containing the entry point specified beginning at symbolic location BAL13. That module will be loaded into the program partition in which the program is currently executing, at the relative load address specified by the Linkage Editor. Control will be transferred to the newly loaded module at the entry point specified at BAL13. Error completion processing will be handled by the program. This macro call will generate both the Service Request and the macro expansion in-line.

## **LOAD MACRO**

The LOAD macro transfers the program load module specified in the macro, or containing the entry point designated in the macro, into the program partition of main storage. The module is loaded at the relative load address specified either by the Linkage Editor or by the macro. Control is returned to the point of call after the LOAD is completed or immediately after the macro is accepted by the system. The address of the primary entry point of the newly loaded module or of the named entry point is returned in the SR packet. If RETURN=YES is coded, the problem program must check the completion status indicator to determine when the LOAD is completed so that the entry point address can be referenced.

The LOAD macro is used primarily for the following purposes:

- To bring fixed data modules, such as translation tables or prepared messages, into dynamically variable storage or overlay areas.
- To load a program segment at the address specified by the Linkage Editor and transfer control at some other point in the problem program. The user must, of course, code the instructions to transfer control to the loaded program.

Any relocatable references to a module that is loaded at a relative address other than that assigned by the Linkage Editor are invalid.

The content of the LOAD macro is as follows:

```
[label]    LOAD    { MOD=symbolic location  
                  of module name  
                  or  
                  ENTRY=symbolic location  
                  of entry point  
                  [ ,LOADADR=symbolic location  
                  of load address ]  
                  [ ,ERRCOMP= { YES }  
                  [ NO ] ]  
                  [ ,LIST= { YES }  
                  [ NO ] ]  
                  [ ,RETURN= { YES }  
                  [ NO ] ]
```

MOD= specifies the start address (symbolic location) of a 10-byte field, of which the first 8 bytes contain the EBCDIC name of the module to be loaded. The last two bytes will receive the primary entry point returned by the Loader. MOD= is required for LOAD by module name only, and precludes the use of the ENTRY keyword.

ENTRY= specifies the start address (symbolic location) of a 10-byte field, of which the first 8 bytes contain the EBCDIC name of the entry point which must reside in one of the defined segments or overlays of the program making the call. The last two bytes of the area will receive the named entry-point address returned by the loader. Use of the ENTRY keyword precludes use of the MOD keyword.

LOADADR= designates the main storage address (symbolic location) at which the requested module is to be loaded. Whenever this keyword-operand is omitted, the requested module will be loaded at the relative address originally specified by the Linkage Editor.

ERRCOMP=YES specifies that control is to be returned to the requesting program if the service request macro completes with errors. ERRCOMP=NO specifies that control is to be retained by the system in the event of an error, and results in program abort. This keyword is optional; the default is NO. Error completion codes are shown in the macro expansion, Appendix F.

LIST= controls generation of the Service Request and of the parameter string for the macro. YES generates an object string for the macro, but no SR instruction. General register 6 must contain the address of the parameter string when the program is executed. NO generates an SR instruction with no parameter string. Omission of the LIST keyword generates an SR instruction with the macro expansion (parameter string) in line, immediately following the SR.

RETURN=YES specifies that control is to be returned to the point of call immediately after the LOAD request is recognized by the system and queued. RETURN=NO results in return of control only after the LOAD macro has completed processing, and the proper module is loaded. The problem program is placed in a wait state until completion. The default is NO.

The address of the proper entry point will be returned with the packet upon completion of the request. If RETURN=YES was coded, the problem program is responsible for checking the completion indicator (CI) bit in the packet to verify completion of the request.

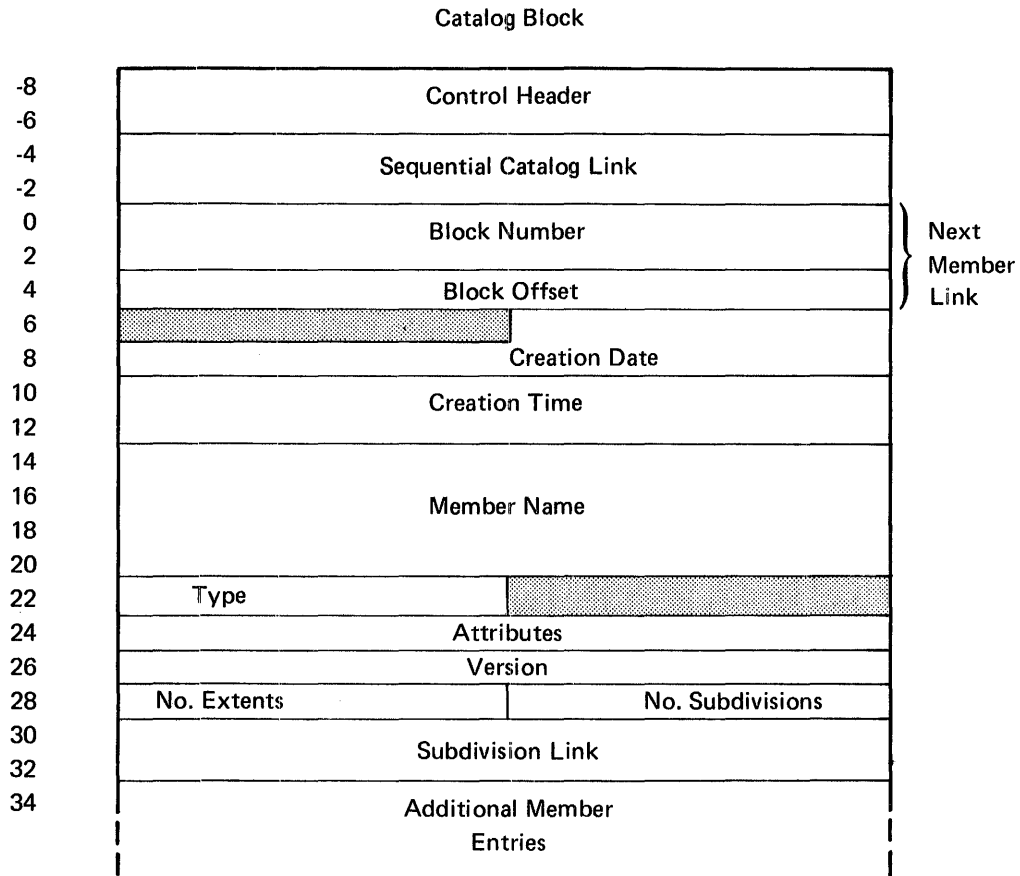
#### **SAMPLE LOAD MACRO**

The following is an example of a LOAD macro.

```
LOAD    MOD=PROG16A,RETURN=NO,LIST=NO,LOADADR=CATT
```

In this example the private library (if any) and \$SYSLODLIB will be searched for a module named in the field whose symbolic address is PROG16A. The module will be loaded at symbolic location CATT. Error processing will be handled by the system. This macro will generate only the Service Request in-line. Another LOAD macro in the problem program must set up the parameter list and the problem program must load general register 6 with the address of the parameter list prior to execution of the macro illustrated here. Control will be returned after the request has been completed and the module has been loaded. This macro has no label.

# CATALOG BLOCK



<u>Name</u>	<u>Bytes</u>	<u>Description</u>
Control Header	-8 -- -5	Common stored data format standard record header.
Sequential Catalog Link	-4 -- -1	Points to next catalog block in this library; last link is zero.
Next Member Link	0-5	
Block Number	0-3	The associated catalog block number which contains the next member entry in the chain.
Block Offset	4-5	The byte displacement within the block.
	6	Reserved for system use.
Creation Date	7-9	Date member enters the library; form yyjjj, packed decimal: <div style="margin-left: 20px;">yy    year            jjj    Julian date</div>

<u>Name</u>	<u>Bytes</u>	<u>Description</u>
Creation Time	10-13	Time member enters the library; form hhmmss, packed decimal: hh Hour mm Minute ss Second
Member Name	14-21	Member identification; 1-8 alphanumeric characters, left justified, blank filled.
Type	22	Code which indicates the type of member: Bit 0 Source member Bit 1 Unused Bit 2 Object member Bit 3 Absolute load member Bit 4 Relocatable load member Bit 5 Macro member Bit 6 Procedure member Bit 7 Unused
	23	Reserved for system use.
Attributes	24-25	Code to define characteristics which are unique within each type.
Version	26-27	Optional version identifier for the member.
No. Extents	28	Number of user words which may be attached to the entry; range 1-10.
No. Subdivisions	29	Number of subdivision descriptors contained in this entry; range 1-5.
Subdivision Link	30-33	Initial block number of subdivision.

# MEMBER DEFINITION BLOCK (MDB) FOR LOAD MODULES

Member Definition Block for Load Modules

0	MDB Switches		
2	MDB Size		
4	Member Name		
6	Member Name		
8	Member Name		
10	Member Name		
12	Type	Reserved	
14	Attributes		
16	Version		
18	Extents (=6)	Subdivisions (=3)	
20	CEPL Subdivision Block No.		
22	CEPL Subdivision Block No.		
24	TEXT Subdivision Block No.		
26	TEXT Subdivision Block No.		
28	RCS Subdivision Block No.		
30	RCS Subdivision Block No.		
32	Load Module Size (bytes)		} Available user extension bytes at time member is stored
34	Reserved	Tag	
36	Relative Load Address		
38	Primary Entry Point		
40	FDT Byte Offset		
42	Total Size Commitment (bytes)		

<u>Name</u>	<u>Bytes</u>	<u>Bits</u>	<u>Description</u>
MDB Switches	0	0	Indicates member could not be found in the library search.
	0	1	Member found but store was requested in an ADD mode.
	0	2	I/O error during library function.
	0	3	Reserved for system use.
	0	4	0 indicates REPLACEMENT mode store. 1 indicates ADD mode store.
	0	5	Delete member when found.
	0	6-7	Reserved for system use.
	1	0-7	Reserved for system use.

<u>Name</u>	<u>Bytes</u>	<u>Bits</u>	<u>Description</u>
MDB Size	2-3	0-7	Length of MDB in bytes, not including MDB Switches or Size cells.
Member Name	4-11	0-7	Member identification; 1-8 alphanumeric characters, left justified, blank filled.
Type	12	0-7	Code which indicates the type of member: Bit 0 Source member Bit 1 Unused Bit 2 Object member Bit 3 Absolute load member Bit 4 Relocatable load member Bit 5 Macro member Bit 6 Procedure member Bit 7 Unused
Reserved	13	0-7	Reserved for system use.
Attributes	14-15	0-7	Code to define the characteristics unique to each type; bit 0 indicates member deleted.
Version	16-17	0-7	Optional version identifier for the member.
Extents	18	0-7	Number of user words which may be attached to the entry; maximum = 6 words.
Subdivisions	19	0-7	Number of subdivision descriptors which are contained in this entry.  Maximum = 4 subdivision descriptors.
CEPL Subdivision Block No.	20-23	0-7	Initial block number of the Composite Entry Point List subdivision.
TEXT Subdivision Block No.	24-27	0-7	Initial block number of the TEXT subdivision.
RCS Subdivision Block No.	28-31	0-7	Initial block number of the Relocation Control Stream subdivision.
Load Module Size	32-33	0-7	Size in bytes of the load module.
Reserved	34	0-7	Reserved for system use.
Tag	35	0-7	Reserved for future use for extended addressing.

## B. LIBRARIAN EXECUTION-TIME ERROR MESSAGES

There are two types of SYSOUT error messages: those issued directly by the Librarian (LIBUTIL) program, and those issued directly from the system message library.

### MESSAGES ISSUED BY THE LIBUTIL PROGRAM

The LIBUTIL program execution-time error messages are all printed on the device specified by the DEV= parameter on the //DEF statement that reads //DEF ID=LIST, DEV=. All message error codes begin in print position 2. They have the following fields:

pp	ss	eee	t
----	----	-----	---

- Where:
- pp is always LB, specifying the error as a LIBUTIL error.
  - ss is either ER or WA, where ER specifies fatal errors and WA specifies warning errors.
  - eee is a 3-digit error number specifying the error within the type (ER or WA).
  - t is a single digit which is either 2 to specify warning or 8 to specify fatal error conditions.

After the error code, the following text appears for all messages having the ER specification in the ss field:

#### LIBRARIAN ERROR CODE

The following text appears after all the error codes have the WA specification in the ss field:

#### LIBRARIAN WARNING CODE

For a description of the error code, refer to the explanation of error codes listed below.

ERROR CODE	EXPLANATION OF ERROR CODE
LBER0018	An invalid or unsupported parameter has been specified.
LBER0028	The number of MEM parameters exceeded the maximum.
LBER0038	An invalid or unsupported command is specified.



ERROR CODE	EXPLANATION OF ERROR CODE
LBER0048	The member type in the MEM parameter is invalid.
LBER0058	An FDT could not be found for a required file ident that should have been specified.
LBER0068	The member to be patched was not relocatable (REL) or absolute (ABS).
LBER0078	The patch directives operator is other than VER or REP and is not supported.
LBER0088	The input member that was requested to be printed or punched has a first segment link of zero where the link to the source file should be.
LBER0098	The input member that was requested to be printed or punched cannot be found on ILIB by library search.
LBER0108	The input member that was requested dumped cannot be found on ILIB by library search.
LBER0118	The load input member identification card specifies a segment number that is greater than the maximum segment number for this library.
LBER0128	The load input member identification card is out of place.
LBER0138	Input/output error.
LBER0148	A segment in the data input to be loaded is greater than the highest segment for the present member as specified by the member identification card.
LBER0158	A segment specified in the data input to be loaded is a duplicate of the member being loaded.
LBER0168	No patch is in the patch directive.
LBER0178	An invalid hexadecimal digit is in the patch directive.
LBER0188	A patch verification directive failed to compare equally with the specified relocation attribute.
LBER0198	The input member in an inclusive copy cannot be found on ILIB by library search.
LBER0208	The output member in an inclusive copy was found on OLIB by library search and is protected by the MEM parameter.
LBER0218	The member that was requested patched cannot be found in the ULIB by library search.
LBER0228	In a patch directive, the displacement was to an odd address.

ERROR CODE	EXPLANATION OF ERROR CODE
LBER0238	In a patch directive, the relocation attribute is invalid.
LBER0248	A patch verification directive failed to compare equally with the specified text.
LBER0258	In an update function, the input member cannot be found on the ILIB by library search.
LBER0268	In an update function, the output member specified was found on the ILIB by library search and is protected by the MEM parameter.
LBER0278	The copy member requested during an update function could not be found by library search.
LBER0288	An insert was requested during an update but no input member was specified by the MEM parameter.
LBER0298	The Mth parameter in an update insert or copy directive is less than the Nth parameter in that same directive.
LBER0308	The Nth parameter in an update insert directive is less than the present record position.
LBER0318	During an update, an insert or copy directive exceeded the file size. The N or M was greater than the last record number on that particular file.
LBER0328	The input member that was to be deleted cannot be found by the library search.
LBER0338	The input member to be renamed cannot be found on the ULIB by the library search.
LBER0348	The output member name, the name which the input member name will be renamed to, already exists in the ULIB and is protected by the MEM parameter.
LBER0358	The first segment link in the member to be updated, the input member, is zero. It should contain the relative block of the beginning of the source segment.
LBER0368	An invalid patch directive has been specified.
LBER0378	An invalid member type code is in an existing library.
LBER0388	A patch directive displacement has exceeded the member size.
LBER0398	The number of sub-parameters exceeds the maximum.
LBER0408	The insert directive in the update function has an invalid N specified, either: <ul style="list-style-type: none"> <li>o N = 0 or unspecified which has no meaning in an insert directive,</li> <li>o N = 1 and the input member position is past 1.</li> </ul>

ERROR CODE	EXPLANATION OF ERROR CODE
LBER0418	A sub-parameter length exceeds the maximum.
LBER0428	A right parenthesis is missing from a sub-parameter specification.
LBER0438	A PAR card was not ended by a blank or comma.
LBER0448	A parameter length exceeds the maximum.
LBER0458	A literal string length exceeds the maximum.
LBER0468	A right-most quote in a literal string is missing.
LBER0478	The keyword length exceeds the maximum.
LBER0488	An equal sign is missing in the keyword scan.
LBER0498	An invalid SORTKEY parameter is specified.
LBER0508	An invalid SELECT parameter is specified.
LBER0518	An invalid MODE parameter is specified.
LBER0528	An invalid SEQCHK parameter is specified.
LBER0538	An invalid LIST parameter is specified.
LBER0548	Alphanumeric data was found in all numeric parameters.
LBER0558	An invalid SPACE parameter is specified.
LBER0568	The MEM parameter does not contain a member name.
LBER0578	A sequence step-down has occurred in the specified field when a sequence check was requested.
LBER0588	The sequence field length plus the field position is greater than the IFIL block size.
LBER0598	The sequence field length exceeds the maximum.
LBER0608	The ILIB block size is not equal to the OLIB block and a copy is requested.
LBER0618	The member type specified is not compatible with the library function requested.
LBER0628	MODE=F but the member type is not ABS, or the member type is ABS, but the sub-division two (text) is zero.
LBER0638	Multiple members have been specified for the UPDATE function.

ERROR CODE	EXPLANATION OF ERROR CODE
LBBER0648	A type 1 member. MEM=(input-member,member-type) is specified and is illegal. UPDATE needs an output member name.
LBBER0658	No member is specified for the UPDATE function.
LBBER0668	A library block size exceeds the maximum buffer size.
LBBER0678	Null input to update create mode.
LBBER0688	Invalid UMODE specification.
LBBER0698	N, M of updated directive is not sequential.
LBBER0708	N or M of update directive exceeds length of sequence field specified in SEQPOS.
LBBER0718	Null input to load function.
LBBER0728	The OLIB type parameter is invalid (i.e., is not SYM, ENC or ALL).
LBBER0738	The numeric parameter is larger than 5 digits.
LBBER0748	The output library block SIZE= parameter is less than the minimum 84 bytes.
LBBER0758	The library type is invalid on an existing library.
LBWA0012	Excessive parameters were specified and ignored.
LBWA0022	An inconsistent sequence type parameter was specified.
LBWA0032	An inconsistent list type parameter was specified.
LBWA0042	Duplicate specifications of INITPG.
LBWA0052	Duplicate specifications of LIST.
LBWA0062	Duplicate specifications of MODE.
LBWA0072	Duplicate specifications of MTYPE.
LBWA0082	Duplicate specifications of NEWSEQ.
LBWA0092	Duplicate specifications of OFIL.
LBWA0102	Duplicate specifications of OLIB.
LBWA0112	Duplicate specifications of PAGESIZ.
LBWA0122	Duplicate specifications of SELECT.

ERROR CODE	EXPLANATION OF ERROR CODE
LBWA0132	Duplicate specifications of SEQCHK.
LBWA0142	Duplicate specifications of SEQPOS.
LBWA0152	Duplicate specifications of SORTKEY.
LBWA0162	Duplicate specifications of SPACE.
LBWA0172	Duplicate specifications of TITLE.
LBWA0182	Duplicate specifications of ULIB.
LBWA0192	Duplicate specifications of VERSION.
LBWA0202	Duplicate specifications of WLIB.
LBWA0212	Duplicate specifications of COMMAND.
LBWA0222	Duplicate specifications of IFIL.
LBWA0232	Duplicate specifications of ILIB.
LBWA0242	SELECT=I and no members supplied. SELECT=E is defaulted too.
LBWA0252	Plus or minus sign in position 1 of update directive assumed to be data.
LBWA0262	Duplicate specification of UMODE.

### MESSAGES ISSUED FROM THE SYSTEM MESSAGE FILE

The following messages can appear on the SYSOUT file; they are issued from the system message file \$MSGLIB. Each message is preceded by three asterisks, a 4-digit, system-oriented hexadecimal status code and an 8-character error code that has the following format:

pp	ss	eee	t
----	----	-----	---

- Where:
- pp is always LB, specifying the error as a LIBUTIL error.
  - ss is variously OP, TR, ST, or SE specifying the module within the LIBUTIL program that issued the message.
  - eee is a 3-digit number 001 through 006.
  - t is a single digit which is always 8 meaning that all the errors are fatal errors.

The message text follows the error code. The text of the message ends with four asterisks.

MESSAGE ID	HEX STATUS COMPLETION CODE	ERROR CODE	MESSAGE TEXT
***	2F01	LBOP0018	<p>INVALID LIBRARY DEFINITION****</p> <p>One of the following conditions has been detected:</p> <ul style="list-style-type: none"> <li>● The file is not sequential.</li> <li>● The library index block is not in the proper format.</li> <li>● The library type does not match the type specified in the library OPEN packet.</li> </ul>
***	2F02	LBOP0028	<p>THE FDT FOR THIS LIBRARY COULD NOT BE FOUND IN THE FDT CHAIN****</p> <p>Either the system failed to open the library or the partition was destroyed.</p>
***	2F03	LBOP0038	<p>INCONSISTENT USAGE SPECIFICATION IN FDT****</p> <p>One of the following conditions has been detected:</p> <ul style="list-style-type: none"> <li>● Library was opened with undefined USAGE= keyword specified.</li> <li>● Library has been opened for I/O.</li> <li>● Library has been opened for input and HBW=0.</li> </ul>
***	2F04	LBTR0048	<p>I/O ERROR ON LIBRARY****</p> <p>A disc I/O error has occurred on a library.</p>
***	2F05	LBST0058	<p>END OF ALLOCATION REACHED DURING ATTEMPTED STORING OF A MEMBER ENTRY****</p> <p>End of disc allocation has occurred for this library.</p>
***	2F06	LBSE0068	<p>MEMBER TYPE, FOUND IN MDB FOR SEARCH OR STORE, IS INVALID FOR THE SUBJECT LIBRARY****</p> <p>The member type requested, searched, or stored is not compatible with the subject library. For example, a SRC (source) member was requested, searched, or stored in a library whose type was ENC (encoded) only. If a SRC member was requested, searched, or stored in a SYM (symbolic) or ALL type library, no error would have occurred.</p>



# INDEX

Absolute load module	1-1,3-8	Identification card, member	2-30
Block size constraints	2-2	IFIL keyword	2-12
BOUND keyword	3-15	ILIB keyword	2-6
Bound register	3-15	Index table	A-1
		INITPG keyword	2-10
Catalog block	2-1,A-3	Keyword-operands	
Catalog ordinals	A-1	for LIBUTIL	2-6
Checkout debugging	4-1	for Linkage Editor	3-10
Coding	1-1	Library	
COMMAND keyword	2-6	description	2-1
Common allocation	3-22	structure	2-1
Compilation	1-1	Library block size	2-2
Composite entry point list	A-8	Library compression	2-23
Composite length	3-32	Library directory	2-1
Control Language requirements		Library macros	G-1
for Librarian	2-4	Library member	1-2,2-1
for Linkage Editor	3-9	Library member protection	2-8
COPY command	2-19	Library overhead	
Create symbolic member	2-41	source libraries	5-1
		encoded libraries	5-2
Data separator	2-4	Library search domain	3-4
DELETE command	2-21	Library structure	2-1
Directive set	3-3	Library types	2-7
Directives, linkage editor	3-15	Library utility	2-4
Directory, library	2-2	LIBUTIL	2-4
DUMP command	2-29	LIBUTIL keyword summary	2-14
Dumped output format	2-11	Linkage Editor	
Duplicate entry points	3-4	general	1-1
		description	3-1
END directive	3-18	Linkage editor input	3-3
ENTRY directive	3-16	Linkage editor output	3-7
Entry point list	A-6,D-1	Link-edit map	3-27
Entry point search domain		List file, linkage editor	3-9
description	3-4	LIST operand, LIBUTIL	2-9
specified by USE directive	3-17	LOAD command	2-32
ERROR keyword	3-14	Loader	4-1
Error messages		Loader macro expansion	F-1
for Librarian	B-1	LOAD macro	4-2
for Linkage Editor	C-1	Load module	
for Loader	H-1	absolute	1-1,3-8
Executable program	1-1	relocatable	1-1,3-8
Expressions in SEG directives	3-19	LSD keyword	3-11
External references	3-4	LST keyword	3-13
FETCH macro	4-2		



MDB layout	A-5,A-7	Redefining primary entry points	3-16
Member definition block		Relocatable load module	1-1,3-8
for load modules	A-7	Relocatable object module	1-1
for object modules	A-5	Relocation control stream	A-8
Member identification card	2-30	Relocation group dictionary	A-6,D-4
Member, library	1-2,2-1	RE:NAME command	2-26
Member separator card	2-30	Renaming library members	
Member type		with COPY command	2-19
in catalog block	A-4	with PACK command	2-25
in MDB	A-6,A-9	with RENAME command	2-26
mixing on libraries	2-2		
valid codes	2-8	SEG directive	3-18
MEM keyword	2-7	SEG terms	
Memory occupation	3-19	control section name	3-21
Mixing member types	2-2	library name	3-22
MODE keyword	2-11	module name	3-21
Modify symbolic member	2-41	SELECT keyword	2-9
Module linking	3-1	Separator card	
MTYPE keyword	2-8	data	2-4
		member	2-30
NAME directive	3-16	SEQCHK keyword	2-11
NEWSEQ keyword	2-11	SEQPOS keyword	2-10
		Sequence field	2-10
Object modules	3-5	Service request expansion	E-1
Object module structure	D-1	SIZE keyword	3-14
OFFSET keyword	3-12	Source module	1-1
OFIL keyword	2-12	SPACE keyword	2-10
OLIB keyword	2-6	SRH keyword	3-13
ORG keyword	3-12	Standard subroutine use	3-1
Operators in SEG directives	3-19		
		Text string group	D-2
PACK command	2-23	TITLE keyword	2-9
PATCH command	2-34		
PGM keyword	3-11	ULIB keyword	2-12
PGSIZE keyword	2-10	UMODE keyword	2-12
POOLSIZ		UPDATE command	2-41
keyword	3-13	UPDATE directives	
use in MDB	A-10	pointer directives	2-43
Presetting common	3-23	copy directives	2-47
Primary input file	2-12	USE directive	3-17
Primary input module	3-3		
PRINT command	2-38	VERSION keyword	2-10
PRIV keyword	3-15		
Privileged task	3-15	WLIB keyword	2-13
Program generation	1-1	Work library	2-13
Protection of member	2-8		
PTOC		XQT keyword	3-11
command	2-13		
sample listing	2-16		
PUNCH command	2-39		
Punch symbolic member	2-39		
Punch encoded member	2-29		