MRX/OS Program Library Services

Reference Manual

2200.005

MEMOREX

Computer System Products

March 1973 Edition

Requests for copies of Memorex publications should be made to your Memorex representative or to the Memorex branch office serving your locality.

A reader's comment form is provided at the back of this publication. If the form has been removed, comments may be addressed to the Memorex Corporation, Publications Dept., 8941 – 10th Ave. No. (Golden Valley) Minneapolis, Minnesota 55427.

PREFACE

Program Library Services encompass three main system products, the Librarian utility program (LIBUTIL), the Linkage Editor, and the Relocating Program Loader. Each of these system programs is discussed in a separate section of this manual. Control Language statements used to request the programs are designated, and the //PAR cards for LIBUTIL, directives for the Linkage Editor, and macros for the Relocating Program Loader are discussed in detail.

Job stream examples are supplied with each product description. Library overhead is discussed in Section 5. Table formats and error codes are specified in the appendices.

Other Memorex manuals with which the user should be familiar are:

MRX/OS Control Program and Data Management Services, Extended Reference

MRX/OS Control Language Services, Extended Reference

MRX/OS Assembler Reference

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

All programs intended to execute under the Memorex Operating System must be processed by the Linkage Editor, must be entered on libraries, and must be loaded by the Relocating Program Loader. The relationship of these programs to each other and to the rest of the operating system can best be described by defining their roles in program generation. Program generation comprises five basic steps, each of which produces a uniquely structured physical module. A module in any of the five forms may be an entire program or a part of a program, such as a subroutine or overlay. The five basic steps of program generation are coding, compilation, linkage-editing, library processing, and loading.

CODING

The coding process results in a collection of statements in whatever programming language is used. This collection of statements, when named, is called a source module. It is acceptable as input to either the library utility UPDATE function or the intended language processor.

COMPILATION

Compilation is the process of translating a source module into an intermediate form of machine language and codes that allow linking these modules together. These modules, which are the output of all language processors, are called relocatable object modules.

LINKAGE EDITING

The Linkage Editor creates loadable, executable programs, using as input one or more object modules and/or relocatable load modules. The Linkage Editor resolves all cross references between the modules and combines them into one or more load modules. An executable program consists of one or more load modules. A load module is that portion of an executable program that is processed and loaded by a single request to the Relocating Program Loader. The number of load modules comprising an executable program depends entirely upon the overlay structure specified by the programmer. Two types of load modules can be produced.

- Absolute load modules are link-edited to execute at a fixed main storage location, and will always be loaded at that location. They are portions of the Operating System Control Programs and Transient Service Routines.

- Relocatable load modules are link-edited to execute at a base of relative zero and may be loaded and executed at any main storage location. At load time, the Relocating Program Loader modifies all address references to reflect actual load addresses.

LIBRARY PROCESSING

The Linkage Editor calls the Librarian to catalog and write load modules on user-specified libraries. The libraries are partitioned sequential data files; a library may contain a variable number of "members", each cataloged by name and type. Members may be load modules, unlinked object modules, source language programs, procedures, macros, or other types of data. The Librarian provides routines that allow the user to perform library maintenance functions, such as patching load modules, copying members from one library to another, modifying source members and so on.

LOADING

The Relocating Program Loader retrieves load modules from a user-specified library. The modules are loaded into main storage at the proper location for execution, and address references are modified as required.

2. LIBRARIAN

INTRODUCTION

A library in the context of this manual is a partitioned data file that consists of a group of program modules, procedures, or other types of data, called members. The Librarian is a collection of routines that support the creation, maintenance, and processing of libraries. These routines allow the system to open, search, and close libraries and to store members in libraries. They also provide a utility program (LIBUTIL) that enables customer programmers to perform various library maintenance functions such as adding members to libraries, modifying existing members, deleting members, copying, dumping, printing, and punching members of a library. The LIBUTIL program executes in a user partition of 8192 bytes, or larger.

LIBRARY DESCRIPTION

Libraries are sequentially organized and occupy space on a single disc volume. The space need not consist of contiguous tracks. Space for a library is allocated through Control Language //DEFINE statements. Unlike other sequential files, space for a library cannot be expanded after allocation. Six types of library members are supported, divided into two categories: symbolic and encoded. The first category, called symbolic, includes the following members:

- Source input to a language processor (assembler or compiler)
- Unassembled macros
- Control Language procedures

The second category, called encoded, includes these members:

- Unlinked object modules
- Relocatable load modules
- Absolute load modules

LIBRARY STRUCTURE

Along with the data for each member, each library includes a directory that contains the identification and location of each member in the library as well as descriptive information about each member.

Members are added sequentially to the end of a library. When no more space is available, no additional members can be added. However, a member that is no longer needed may be deleted from a library; its directory entry is flagged to indicate that the member is inactive.

The space occupied by the deleted member and its directory entry does not become available until the library is compressed.

The library directory consists of a catalog ordinal table and a catalog. The catalog ordinal table occupies the first blocks of the library and contains information about the library, including the type of library, and pointers to the current catalog block, and the highest block written. The catalog consists of one or more blocks that contain member entries. Each entry includes the member name, version, type, date and time of creation, and starting locations of its data subdivisions. Member entries may vary in length, but all catalog blocks are the same length as the allocated block size of the file. Therefore the number of member entries per catalog block is variable. As each member enters the library, the Librarian constructs its catalog entry in the current catalog. After a member has been stored, and its member entry made in the catalog, if there is not enough space in the current catalog block for the maximum sized member entry, a new catalog block is built into the next available area in the library. Thus the member catalog occupies only the space required for current members. The data for each member follows the catalog block in which the member entry appears. Figure 2-1 illustrates the structure of a library. Appendix A describes the catalog ordinal table and the catalog in detail.

DATA STRUCTURE

All blocks in a library are the same length as specified when the file is allocated. Other system products impose certain restrictions on block size, according to the type of library.

- Encoded libraries require a fixed block size of 256 bytes

- Symbolic libraries must have a block size of at least 84 bytes, but symbolic libraries used for storing Control Language procedures must not exceed a block size of 128 bytes. Block size for symbolic libraries used for source members depends on the requirements of the particular language processor. On the minimum system, such libraries must have a block size of 84 bytes for assemblers and compilers.

Different types of members can be mixed on the same library. However, the user must exercise care when doing so, because of the size constraints and Control Language restrictions (see *Programming Considerations* in this section). For example, the LIBUTIL program does not prohibit placing a Control Language procedure member on a library whose block size exceeds 128 bytes, but the Control Language Services cannot retrieve a procedure from such a library.

The data formats used in constructing libraries vary according to the member type. Some members use a four byte record header (not common stored data format) while others are without headers.

Each module or subdivision of a module terminates with a data block written with a zero length data field (EOF indicator).

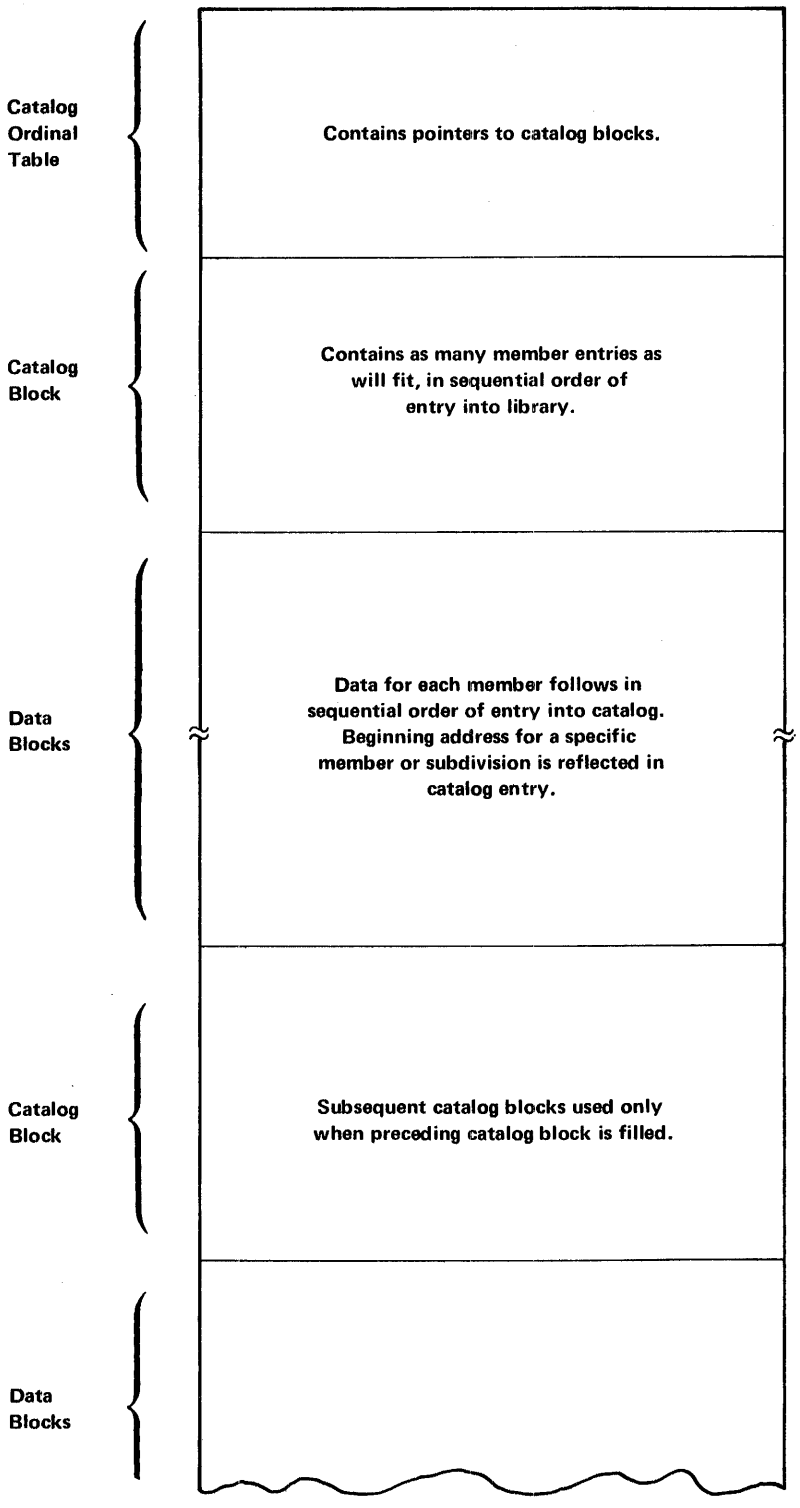


Figure 2-1. Library Structure

LIBRARY UTILITY PROGRAM (LIBUTIL)

The library utility program (LIBUTIL) provides commands for the creation and maintenance of libraries. It is invoked as a separate job step through Control Language Services and executes in a program partition of 8K bytes.

CONTROL LANGUAGE REQUIREMENTS

A single multi-functioned program, LIBUTIL, is specified as the operand for the PGM= keyword on the //EXECUTE statement. LIBUTIL commands and all other parameters are submitted in keyword-operand form on a //PAR statement. LIBUTIL allows three formats for the keyword-operands:

```
keyword=operand
keyword=(operand1, . . . ,operandn)
keyword='literal string'
```

Parameters must be separated from each other by a comma, and no parameter may be divided between two //PAR cards. When two or more //PAR statements are included for a single command, a comma must follow the last parameter on all but the last card. When only the first operand of a group within parentheses is specified, the parentheses may be omitted. Any omitted operands prior to the last specified within parentheses must be represented by commas. For example, KEYWORD=(,, operand3,operand4). In this manual, optional parameters are denoted by brackets, [ ]. When a choice of operands is available, they are enclosed in braces, { }

Many //PAR statements requesting separate commands may be stacked to perform multiple functions in a single job step. Although there are some restrictions on the number of keywords allowed for certain functions, there are no restrictions on the number or combinations of functions that may be requested. Similarly, there are no restrictions on the number of libraries referenced by a single step. Each command is treated as a logically independent substep by the Librarian, its libraries opened and closed in the substep. The ILIB and OLIB keywords, described under the heading *Keyword-Operand Descriptions* in following text facilitate specifying many libraries within the same step.

DATA SEPARATOR STATEMENT

Library utility commands can be stacked in the same job step. In addition, data streams for the commands can use distinct data files or can be stacked behind a single //DATA card. The data separator statement enables the library utility to determine the end of data supplied for each command. The format of this statement is as follows, beginning in column one:

```
/*LIB
```

Sets of data stacked behind a single //DATA statement must be in the same order as the commands with which they are associated. Each set except the last must be terminated by the /*LIB statement. The last set does not require a /*LIB statement, but it is allowed. The /*LIB statement is applicable to the LOAD, PATCH, and UPDATE commands.

Example:

```
//JOB   NAM=SAMPLE
//EX    PGM=LIBUTIL
//DEF   ID=LIST,DEV=PRT
//DEF   ID=SEQIN,FIL=CHANGES
        •
(Other required //DEF cards)
        •
//PAR   COM=LOAD, . . .
//PAR   COM=PATCH,MEM=(XYZ,REL), . . .
//PAR   COM=PATCH,MEM=(ABC,REL), . . .
//PAR   COM=UPDATE, . . .
        •
(Other optional //PAR cards)
        •
//DATA FIL=CHANGES
        •
(Member decks for loading)
        •
/*LIB
        •
(Patch directives for member XYZ)
        •
/*LIB
        •
(Patch directives for member ABC)
        •
/*LIB
        •
(Directives and/or data for UPDATE)
        •
/*LIB
/*
//EOJ
```

PROGRAMMING CONSIDERATIONS

Although the librarian routines themselves are general-purpose, the manner in which a library is to be used in the system may impose some restraints. For example, libraries containing Control Language procedures or system macros must be cataloged on the system central catalog when they are allocated, and must be mounted on drives declared to be shared at execution time. Other libraries may be cataloged or uncataloged and may be

mounted on shared or unshared devices. Control Language //DEFINE statements for ID=$LODLIB are required for execution of programs on uncataloged load libraries but no //DEF statement is required for those on cataloged load libraries.

KEYWORD-OPERAND DESCRIPTIONS

Each LIBUTIL function, named with the COMMAND keyword, is designed for a specific task. However, additional processing may be included with the function by using optional keyword parameters. For example, when copying a library, partial packing of the library can be accomplished by including or excluding certain members with the SELECT keyword, and renaming can be performed with the MEM keyword. Most keywords have default values, and need be coded only when a value other than the default is required. The following paragraphs describe the uses of the keyword-operands; the LIBUTIL functions are described under the heading *Command Descriptions* later in this section.

FUNCTION KEYWORD (COMMAND OR COM)

The COMMAND= keyword specifies the LIBUTIL function to be performed. COMMAND can be abbreviated to COM. The following commands are recognized by LIBUTIL:

PTOC	RENAME	PRINT
COPY	DUMP	PUNCH
DELETE	LOAD	UPDATE
PACK	PATCH	

This parameter must be specified; there is no default. Each of the functions is described in detail under the heading *Command Descriptions.*

INPUT LIBRARY (ILIB)

The ILIB keyword-operand designates the library to be processed by LIBUTIL as the primary input library. The 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 for this keyword. If omitted, the standard default INPUT is used. This keyword applies to the COPY, DUMP, PTOC, PRINT, PUNCH and UPDATE commands.

The library specified by ILIB must have been initialized by the Librarian as a result of a command employing the OLIB parameter, or as an output library as a result of a call from programs such as the Assembler or Linkage Editor.

OUTPUT LIBRARY (OLIB)

The OLIB keyword-operand specifies the identification and type of output library to be generated or modified by LIBUTIL processing. It has the following format:

```
OLIB=(ident[,type])
```

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:

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

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

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

Keyword ①	Default	Entries by Command										
COMMAND or COM	None ①	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	O②	O②	O	O	O
MEM	None	O③	R④	R④		O③	R④	R④		R④	R⑤	R⑥
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 ⑦	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 ⑧										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 TIME=205423. PAGE: 001
FILE LABEL: $OSRSDNTLIB /ALL

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

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=RA2,
//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={1 / 2 / 3}]

[,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 LIBUTIL 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={1 / 2 / 3}]
[,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:

Column	Contents
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 LIBUTIL 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:

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

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:

$\left\{\begin{matrix}VER\\REP\end{matrix}\right\}$ 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|3}]
[,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)}]
         {NO                       }
[,LIST={YES}]
       {NO }
[,INITPG=initial page number]
[,PGSIZE=lines per page]
[,SPACE={1}]
        {2}
        {3}
[,TITLE='literal string']
```

The default values listed in Table 2-11 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 LIBUTIL 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)
      .
      .
      .
```

The pointer directive can also be used to add data at the beginning of the output library member (before copying anything from the input member). Initially, the internal record pointer precedes the first record of the input member and is not moved until a directive is encountered. (If no directive is encountered before the end of the input data stream, a directive to copy the remainder of the input member is implicit.) Data can be added to the output library member after the last record copied from the input member by specifying a value on the pointer directive equal to the last relative record number in the input member. The following data stream illustrates these two capabilities. Assume the input member contains 50 records and the last five records are to be deleted.

```
//DATA FIL=X
Data item 1   }
      .       }
      .       }   (Add data to beginning of output member)
Data item n   }
- 46,50           (Copy input member through record 45, delete remaining
                  records)
Data item n+1 }
      .       }
      .       }   (Add data to end of output member)
Data item n+m }
/*LIB
```

Pointer by Sequence Number

When sequence number mode is selected (by UMODE=SEQ), the UPDATE program performs the same kinds of copy, insert, and delete operations as it does for relative record number mode, except that a sequence number field is used to identify records. The internal pointer is moved accordingly. Certain basic differences in operation should be noted. The chief difference is that sequence numbers are not necessarily consecutive; that is, the user may have specified an increment greater than one for the sequence field. Therefore, the user must exercise special care when specifying values for the directives.

There is no requirement that sequence numbers specified on the directive statement match any sequence numbers in the input member. The only requirements are that the sequence number field in the input data stream must be in the same position as it appears in the input member and that no sequence number in a directive may be higher than the highest sequence number in the input member. For example, assume an input member with 100 records, sequence beginning at 10 with an increment of 10, to which we want to add two lines of data between sequence numbers 40 and 50, delete sequence number 60, and replace sequence numbers 110 and 120 with one line of data. The input data stream could appear in a number of ways, as illustrated below.

```
//DATA FIL=X                                    //DATA FIL=X
- 40              (Copy first four records)     - 49
Data item 1       (Add data items)              Data item 1
Data item 2                                     Data item 2
- 60,60           (Copy record 50, delete record 60)   - 55,65
- 110,120         (Copy records 70-100, delete  - 102,129
                   records 110, 120)
Data item 3       (Add data item)               Data item 3
/*LIB            (Copy remainder of input member)   /*LIB
```

As can be seen in the example, sequence numbers in the directives need not match those in the input member. The user can specify sequence numbers in a range which includes the sequence numbers affected and obtain the same results as if he named the specific sequence numbers involved.

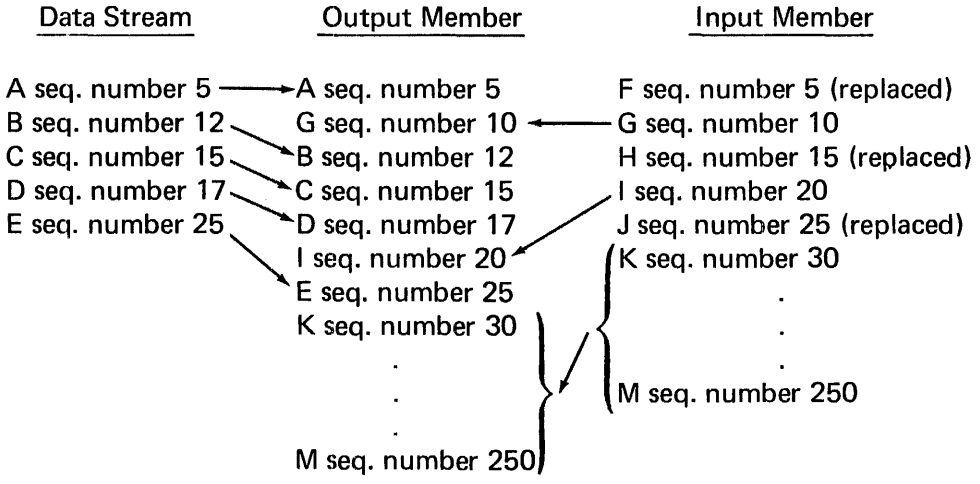
An additional form of the directive can be specified for sequence number mode only. The format is as follows:

```
- *          (Minus sign, space, asterisk)
```

This directive instructs the update program to add the data, in the input data stream following the directive, to the output library member in the proper collating sequence. The program replaces any records in the input member that have matching sequence numbers in the input data stream and copies unmatched records.

To illustrate this capability, assume an input library member of 50 records with a sequence field beginning with 05 and a sequence increment of 5. The following data stream will produce the results shown.

```
//DATA FIL=X
- *
Data seq. number 5
Data seq. number 12
Data seq. number 15
Data seq. number 17
Data seq. number 25
/*LIB
```



The sequence field comparison is active only for the data immediately following the -* directive. Other types of directives can be present after the data, and the new directives can be followed by data that does not have a sequence number.

Copy Directive

The copy directive enables the user to copy entire members, combine two or more members, or move statements from one or more members to another member in the same library. Unlike the pointer directive, no internal position pointer is moved as a result of the directive. However, if a primary input member is specified with the MEM keyword, the implicit directive to copy the remaining records from the position of the internal position pointer through the end of the primary input library member is effective. Therefore if remaining records from the primary input member are not wanted on the output member, a pointer directive deleting the unwanted records must be included before the data separator statement for the command.

The copy directive is identified by a plus sign in column one and a blank in column two. There are three forms of the directive:

- + n,m,input-member-name[,type] [,library-id]
  This form specifies that records from relative record number or sequence number n through m, inclusive, are to be copied from the named input member to the output member named in the MEM= operand. When copying by sequence numbers (UMODE=SEQ), n and m are inclusive bounds. If either does not exist on the input member, the next sequence number higher than n or next sequence number lower than m found on the member are used. The value specified for m must not exceed the highest relative record or sequence number in the input member. Member type is not required if the MTYPE operand is specified. Library-id is the identifier of the library on which input member name is cataloged, and is required when the input member is not on the primary input library specified by ILIB. When type is omitted and library-id is present, the input member name and library-id must be separated by two commas.

- + n,input-member-name[,type] [,library-id]
  This form specifies that relative record or sequence number n through the end of the member are to be copied. Specifications for type and library-id are the same as for the preceding form.

- + input-member-name[,type] [,library-id]
  This form specifies that the entire member is to be copied. The same rules for type and library-id apply as specified for the preceding forms.

Example of use of copy directives:

```
//JOB   NAME=SAMPLE
//EX    PGM=LIBUTIL
//PAR   COMMAND=UPDATE,MEM=(,,NEWONE,SRC),OLIB=(INPUT,SYM)
//PAR   COMMAND=UPDATE,MEM=(NEWONE,SRC,SCNDRY,SRC)
//PAR   COMMAND=UPDATE,MEM=(,,TERTIARY,SRC)
//DEF   ID=LIST,DEV=PRT
//DEF   ID=INPUT,FIL=SRC01,STA=(P,O)
//DEF   ID=OUTPUT,FIL=SRC02,STA=(P,O)
//DEF   ID=SEQIN,FIL=SRCFILE
//DATA FIL=SRCFILE
*       SAMPLE INPUT DATA           CARD 001
*       SAMPLE CONTINUES            CARD 002
*       LAST SAMPLE CARD            CARD 003
/*LIB
*       THIS PROGRAM DEMONSTRATES INPUT     CARD 006
*       TO SOME SOURCE EDITOR               CARD 007
*       INSERT THESE FOUR CARDS             CARD 008
*       AHEAD OF EXISTING CARDS             CARD 009
/*LIB
***     I WISH TO INTRODUCE SOME INFO       CARD 222
***     FROM THE CARD INPUT STREAM          CARD 223
+ 1,4,SCNDRY,SRC,OUTPUT
+ 1,2,NEWONE,SRC,INPUT
**      ADD THIS CARD                       CARD 333
**      ALSO THIS ONE                       CARD 334
+ 3,3,NEWONE,SRC,INPUT
/*
//EOJ
```

The results of this job will be the following:

Member NEWONE	Member SCNDRY	Member TERTIARY
Card 001	Card 006	Card 222
Card 002	Card 007	Card 223
Card 003	Card 008	Card 006
	Card 009	Card 007
	Card 001	Card 008
	Card 002	Card 009
	Card 003	Card 001
		Card 002
		Card 333
		Card 334
		Card 003

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.

The Linkage Editor resolves externals according to the following hierarchy:

1. Entry points of the load module containing the external references are searched first.

2. Modules listed in any specified EPSD of the module containing the external reference are searched. Modules in the domain are searched for the entry point in the order in which their USE directives were included in the directive set.

3. Remaining modules already collected in the link-edit are searched for a matching entry point.

4. The library search domain specified by the LSD keyword is searched.

When a required entry point is not located within the hierarchy above, a field of '****' will be displayed on the cross-reference list entry for the desired external, and the execution will proceed according to the ERROR keyword on the //PAR statement. There is no other notification of this condition other than the asterisks.

OBJECT MODULES

An object module is the output of a single execution of a language processor. It consists of the text, relocation dictionary, and entry point list of one or more Control Sections or Common Blocks. The information required by the Linkage Editor to resolve the cross-references between different object and load modules is contained in the dictionaries and entry point lists. The text consists of the actual instructions and data fields of the object module with embedded relocation information. A fourth type of information, Linkage Editor SEG directives, may also be a part of an object module. These specify the desired physical organization, segmentation, and overlay structure to be used by the Linkage Editor in constructing the output program.

Object modules are always placed in either a permanent library or a temporary "Compile-Link-and-Execute" library, and are physically constructed according to the rules for libraries discussed in Section 2 of this document and the associated appendices.

The text, dictionaries, entry points, and directives are described within the library directory entry for that module as sub-sets. Object module sub-sets are called the Entry Point List, the Text String Group, the Relocation Group Dictionary, and Directives. These are described and illustrated in Appendix D.

LANGUAGE PROCESSOR RELATIONSHIPS

The Linkage Editor produces executable programs from previously constructed load modules and from object modules generated by the language processors. In order to do this, the Linkage Editor makes use of control sections, entry points, external references, and common areas. The following paragraphs define these items in terms of the source language statements used to create them.

ASSEMBLER

In the Assembler language, control section, entry point, external reference and common area correlate directly to Assembler Language statements as described below.

A control section is defined by a CSECT statement, and is delimited by either an END statement or another CSECT statement. (Any assembly of executable code without a CSECT or COM statement is still considered to be a control section with a blank CSECT name. The **MRX/OS Assembler Reference** manual discusses this in detail.)

An entry point is defined with an ENTRY statement.

A primary entry point is specified by the END operand, otherwise it is the first entry point in the object module.

An external reference to a control section or to an entry point is specified with an EXTRN statement.

A common area (common control section or common block) is specified with a COM statement.

The Assembler allows the user to embed SEG directives in his source code and simply copies them to the object modules. All SEG directives must immediately precede the Assembler END statement.

COBOL

A separate control section is produced for the Linkage Section, the Working Storage Section, the Literal Pool, the Display Buffer, one for each File Description, and for the Procedure Division. In a segmented COBOL program, a new control section is produced for each segment created, and the compiler generates a SEG directive. COBOL control sections are always named.

One entry point is generated at the head of the procedure division for each COBOL program.

An external reference is created by the compiler whenever a CALL statement is used, or the need for a MEMOREX-supplied run-time routine is detected.

Object code and/or run time routines may cause common areas to be included in the link-edit map.

FORTRAN

Control sections, entry points, external references, and common areas are specified with FORTRAN source statements, as described below.

A control section is defined with a SUBROUTINE, FUNCTION, or BLOCK DATA statement that specifies the control section name. If the first statement of a FORTRAN routine is not one of these, it is assumed to begin the main routine of the program. This statement automatically defines a control section named either MAIN, or the name specified on the NAME directive for the routine. The control section delimiter is an END statement.

An entry point is defined with an ENTRY statement.

An external reference is created for each EXTERNAL statement, and for each reference (implicit) to a subroutine or function of a user program. Each call to a FORTRAN run-time subroutine generated by the compiler also produces an external reference.

A common area is specified with a COMMON statement. Both named and blank common areas are permitted.

RPG II

The RPG II compiler generates the required control sections, entry points, externals, and common areas required to link-edit an RPG II object program.

A single control section is generated for each RPG II object program. No segmentation is provided in the program.

Object code and/or run time routines may cause common areas to be included in the link-edit map.

An RLABL statement generates an entry point.

An EXIT statement produces an external reference to a subroutine, while a ULABL statement generates an external reference for data. An external reference is also generated when the need for a run-time routine is detected.

Note: Some RPG II run-time routines are themselves segmented. SEG statements specifying their structure are generated in the problem program by the compiler.

OUTPUT

Output generated by the Linkage Editor is of two types. The first is the load module which is always placed in a library as a named member; the second type consists of informational and diagnostic output generated as a sequential data set to be printed.

The load modules generated by the Linkage Editor are either absolute or relocatable. The absolute load module is intended for resident and non-resident portions of the Memorex Operating system (MRX/OS) and for stand-alone utilities and maintenance routines. The relocatable load module is intended to be run in the program partition, under Control Language Services and MRX/OS.

ABSOLUTE LOAD MODULES

An absolute load module is composed of one or two data subsets: the Composite Entry Point List for root modules only, and text for all modules. The text is the actual main storage image of the text including the main storage address at which the module is to be executed. The Linkage Editor will already have added the given load address to relocatable terms.

Absolute Load Modules cannot be link-edited and collected into another program at a later date.

RELOCATABLE LOAD MODULES

A relocatable load module is composed of two or three data subsets. The first subset is a text image of the main storage occupation of the module, except that all references to addresses within the program are relative to an assigned load address of zero.

The second subset is the Composite Entry Point List, contained in root modules only, and includes the entry points of all of the object and load modules that are collected into the program load module. The third subset is the Relocation Control Stream which is a "bit-map" of all the relative addresses that exist in the text, and is one-eighth the length of tha text.

LOAD MODULE CREATION

The Linkage Editor assigns consecutive relative addresses to all control sections and resolves all references between control sections in its processing. Object modules produced by several different language processors may, therefore, be used to form one load module.

Each module to be processed by the Linkage Editor can be made up of one or more control sections, and has an origin that was assigned during assembly, compilation, or a previous execution of the Linkage Editor. In order to produce an executable output load module, the Linkage Editor assigns relative main storage addresses to each control section by assigning an origin to the first control section encountered. Addresses relative to that origin are then assigned to all other control sections to be included in the output load modules. The value assigned as the origin of the control section is used to relocate each address-dependent item in the control section.

The Linkage Editor also resolves external references in input modules. Cross-references between control sections in different modules are symbolic and must be resolved relative to the addresses assigned to the load module. The Linkage Editor calculates the new address of each relocatable expression in the control section and determines the assigned origin of the item to which it refers.

The Linkage Editor calculates composite lengths for each load module and lists them on the link-edit map. The manner in which they are calculated is described in this section under the heading *Executable Program Length* in the discussion of the *Link Edit Map.*

CONTROL LANGUAGE STATEMENT DESCRIPTIONS

The Linkage Editor is requested through Control Language Services and executes as a separate job step in the program partition. The program is specified with the PGM=LNKEDT keyword-operand on the //EXECUTE statement. The remaining keywords of the //EXECUTE statement apply, as described in the Control Language Services, Extended Reference, and are the same as those provided for any problem program.

FILE DEFINITION

All files to be used by the Linkage Editor must be specified on //DEFINE statements in the job step. //DEFINE statements must be included to designate ID=LIST, ID=INPUT, and ID=OUTPUT. In addition, a fourth file, ID=DIR, must be defined whenever a Linkage Editor directive file is used. The following paragraphs describe the four files. Additional //DEFINE statements are required if uncataloged library files are included in the Library Search Domain.

LIST FILE

A list file, ID=LIST, must be specified to receive the printed output produced by the Linkage Editor. The type of listing is specified to the Linkage Editor by the LST keyword on the //PAR statement (see *Parameter Specifications*). This file is in addition to any output that may be printed on the SYSOUT file by the Linkage Editor.

INPUT FILE

An input file, ID=INPUT, must be specified. This file must be the library that contains the primary object module to be link-edited. Other object and load modules to be included in the link-edit may also be included on the library. The LSD keyword on the //PAR statement is used to specify other libraries in which members to be link-edited may be located. Each such library, if uncataloged, must be explicitly defined.

OUTPUT FILE

An output file, ID=OUTPUT, must be specified to receive the output load modules of the link-edit as cataloged entries. Several keywords described in the following discussion of the //PAR statements specify parameters of the output file.

DIRECTIVE FILE

The directive file, ID=DIR, must be defined whenever a sequential file of card images which are Linkage Editor directives is used. This file is called the Directive File and may contain one or more directive sets.

PARAMETER SPECIFICATIONS

The //PAR statements presented to the Linkage Editor contain parameters in keyword-operand form. The format is identical with that used for all other keyword-operands of the Control Language and includes the following forms:

keyword=operand

keyword=(operand1, . . . ,operandn)

The keyword-operands of the //PAR statement specify the primary object module to be link-edited and provide optional processing information to the LNKEDT program. At least one //PAR statement must be included with every execution of the Linkage Editor. Each keyword-operand, except the last one on each //PAR card, is followed by a comma. A keyword-operand cannot be split between two //PAR cards; it must be wholly contained on a single card. Multiple //PAR cards are allowed.

Example:
```
//PAR PGM=ACCT27,PRIV=NO
//PAR LST=XREF
```

The content of the //PAR statement used for the LNKEDT program is:

```
PGM=input name
[,XQT=execute name]
[,LSD={AUTO|(library1[,library2])|NO}]
[,ORG={REL|absolute storage location in hexadecimal}]
[,POOLSIZ=partition space pool size]
[,SRH={YES|NO|ABS}]
[,LST={NORM|XREF}]
[,SIZE=maximum allocation]
[,ERROR={ALL|UNDEF|SIZE|NO}]
[,PRIV={YES|NO}]
[,BOUND={YES|NO}]
[,OFFSET='nnn']
```

The following paragraphs describe these keywords. The keywords may appear in any order on the //PAR statements used. In the event that duplicate keywords are specified, the last one encountered is used.

PGM KEYWORD

This keyword-operand specifies the cataloged name of the primary object module to be link-edited or, if there is a directive set, the name of the directive set as specified on a NAME directive. The operand is an alphanumeric string of 1 to 8 characters, with no embedded blanks or special characters except dash. This keyword is required.

XQT KEYWORD

The XQT keyword-operand specifies the member name under which the main or root load module generated by this execution of the Linkage Editor will be cataloged in the output library (ID=OUTPUT on //DEFINE). This is the name by which the program will be executed (PGM keyword on the //EXECUTE statement).

The operand is a 1- to 8-character alphanumeric string with no embedded blanks or special characters except dash. If the XQT keyword is not supplied, the operand of the PGM keyword on the //PAR statement will be used as the default.

LSD KEYWORD

This keyword-operand specifies how the Linkage Editor will establish the Library Search Domain and hierarchy for locating the load modules required for the current execution. The domain always includes the library specified by ID=INPUT. It is always first in the search hierarchy.

There are three operands which may be used for this keyword. They are AUTO, NO, and a list of one or two library names. When the LSD keyword is not coded, AUTO is assumed as the default.

The AUTO operand specifies that after the primary input library (ID=INPUT on //DEF) has been searched, $SYSOBJLIB, the system object library which contains the subroutines supplied with the operating system, will be searched.

The NO operand specifies that the Linkage Editor will search only the primary input library for externals.

A list of one or two library names may be supplied as the operand of the LSD keyword. When this is done, this list is used in place of $SYSOBJLIB. The libraries are searched in the order listed. The names must be enclosed in parentheses and separated by a comma. For example, LSD=(MAINLIB,RUNLIB).

When uncataloged libraries are included in the list of library names, a //DEFINE card for each such library must be included in the job stream. The //DEFINE card must identify the uncataloged file as ID=LIB1 or ID=LIB2, depending on the position of the uncataloged file in the LSD parameter. For example, if LSD=(MAINLIB,SUBLIB) and SUBLIB is uncataloged, a //DEF card with ID=LIB2 must be present. No //DEF cards are required for these libraries when only cataloged files are included in the list.

If $SYSOBJLIB is to be searched along with another library, it must be included as one of the libraries in the list. Any library name specified on a SEG directive is searched for and must contain the module or CSECT which it modifies. That library must be specified either by ID=INPUT or by listing it as an operand to the LSD keyword. The library specified by ID=INPUT is otherwise always searched before $SYSOBJLIB or other libraries specified as operands to the LSD keyword.

ORG KEYWORD

This keyword specifies whether the load modules generated are to be relocatable or absolute. Either of two operands may be used with this keyword. They are REL or an absolute storage location. When ORG is not coded, REL is assumed as the default.

REL specifies that the Linkage Editor is to generate relocatable load modules which are allocated from a relative address to zero. This is the type of load module which is normally loaded into the program partition.

An absolute storage location is designated as a 1- to 4-digit hexadecimal value enclosed in apostrophes. It specifies that the load modules generated are to be absolute (not relocatable) and are to be allocated from the absolute main storage location specified. This option is intended for use only by the System Generation Procedure when building the operating system, and for stand-alone utilities and maintenance routines.

OFFSET KEYWORD

The OFFSET keyword-operand specifies a hexadecimal value that controls the output of an absolute load module to a library. The specified value is interpreted as a displacement in bytes from the normal beginning of the load module and prevents writing the first n bytes of the load module to the library.

The format of the parameter is as follows:

```
OFFSET='nnnn'
```

'nnnn' is a one to four digit hexadecimal number enclosed in apostrophes and must be an even number, since it represents a word address. This keyword is legal only when generating absolute load modules.

POOLSIZ KEYWORD

The POOLSIZ keyword-operand specifies that the program, when executed, will require only a specified amount of its partition to be available for File Description Tables, TCOM buffers, and Checkout Debugging directives. That amount of space is specified as the operand to this keyword. The operand is a 1- to 4-digit hexadecimal value enclosed in apostrophes.

When POOLSIZ is coded, the amount of space specified by the operand is returned to the partition Space Pool. However, since the Space Pool must start on a 256 byte hardware page boundary, up to 255 bytes more space than was actually specified may be available in the Space Pool. The balance of the partition is available to the program for its use. Exact limits of the area can be determined via the MEMLIM macro.*

If POOLSIZ is not coded, the program is not allowed to reference any unused area of the partition. The Relocating Program Loader will allocate only the space required for the program itself, and will return any excess to the Partition Space Pool for use by system routines.

SRH KEYWORD

This keyword-operand has the effect of modifying the search for entry points. There are three operands for this keyword: YES, NO, and ABS. YES is the default and specifies that the normal search is to use the specification given with the LSD keyword.

NO causes externals to be satisfied only if the modules are specifically included in the map via SEG directives. No library search will be made.

ABS causes only entry points in absolute load modules to be considered in satisfying externals. This feature is intended for systems use only, and should not be coded for problem programs except for stand-alone utilities and maintenance routines.

LST KEYWORD

The LST keyword-operand specifies the characteristics of the printed output (ID=LIST) produced by the Linkage Editor. There are two operands for this keyword, NORM and XREF.

NORM specifies that the normal print format be displayed. This includes the following:

- A header line with a Linkage Editor identification message, time, and date.
- A list of all load modules generated, according to their memory allocation.
- Any errors encountered in processing.

*MEMLIM is described in the **MRX/OS Control Program and Data Management Services, Extended Reference** manual.

When LST is not coded, LST=NORM is assumed as the default.

XREF specifies that in addition to the items listed for NORM, the following will also be displayed.

- A list of all Linkage Editor directives encountered.
- A list of all modules collected as primary and secondary input.
- A list of all load modules generated, according to their memory allocation, and the relative address assignments of all the input modules and control sections.
- A cross-reference list of all externals and entry points as called by, and referenced from, all object modules included in the output program.

SIZE KEYWORD

This keyword-operand specifies the maximum allocation which the output program should require. If the generated output program exceeds this size, an error message will be displayed and the load modules optionally marked as containing an error (see ERROR keyword following).

The operand is a 1- to 4-digit hexadecimal number enclosed in apostrophes. This value must not exceed a value of 65K (decimal). When this keyword is omitted, no checks are made regarding size, except that the size of the total allocation will not exceed 65K decimal.

ERROR KEYWORD

The ERROR keyword-operand specifies the types of errors that will cause the IF code to be set (refer to the //IF statement in the **MRX/OS Control Language Services, Extended Reference** manual). The code will be set to F (EBCDIC) when the error condition specified by ERROR exists.

ALL specifies that any non-fatal error conditions which occur during processing of the program will cause the IF code to be set.

NO specifies that no non-fatal error condition will cause the IF code to be set. Fatal errors prohibit the Linkage Editor from generating an output module.

UNDEF specifies that if an external remains unsatisfied, the IF code will be set.

SIZE specifies that if the program generated has exceeded limits specified by the SIZE parameter, the IF code will be set.

This keyword is designed for use in a link-edit and execute situation. When the user constructs his job stream for such a situation, it is recommended that the code be tested by an //IF statement before attempting to execute the link-edited program.

PRIV KEYWORD

This keyword-operand allows the generated program to be initiated as a privileged task. It allows execution of the privileged instruction set and Class I service requests. There are two operands for this keyword, YES and NO. The default is NO.

YES specifies that the program is being link-edited as a privileged task. This operand is intended for use by system programs, compilers, utilities, and maintenance routines.

NO specifies that the program is link-edited as a non-privileged task and is used for most problem programs. Whenever PRIV=NO is specified or PRIV is not coded, the BOUND=NO keyword-operand is illegal and must not be used.

BOUND KEYWORD

This keyword specifies where the Bound Register is to be set when the program being link-edited is executed. There are two operands for this keyword, YES and NO. The default is YES.

YES specifies that the generated program will run with the Bound Register set to the end of the problem program area. When POOLSIZ is coded this will be the start of the partition space pool as specified.

NO specifies that the generated program will run with the Bound Register set to the end of the partition. This setting does not protect the system tables (FDT's, JCT, and TCT) in the partition. BOUND=NO should be coded only for system programs and other routines which do not depend upon protection of the system tables or which must alter them.

LINKAGE EDITOR DIRECTIVES

The directives are statements introduced as input to the Linkage Editor in symbolic card format. They specify explicit memory allocation of the desired output program, additional input to the Linkage Editor, names and entry points of output load modules, and the modules to be searched in satisfying the externals of a specific input module.

All directives may be introduced as a sequential data file of card images residing on disc (spooled input) in common stored data format and specified by ID=DIR on a //DEF card. The SEG directive only may optionally be included as a part of any object module.

The Linkage Editor directives have the following general format:

```
label      directive      operand1, . . . ,operandn
```

Label specifies a name by which the directive may be referenced. It is a 1- to 8-character alphanumeric string. This field, if present, must begin in column one and must be terminated by a blank. The first character must be alphabetic or a # character. The # in the first character position identifies a CSECT name. Every SEG directive defines a primary resident load module that, if cataloged, is given the name of the SEG statement. Therefore, care must be taken in naming the SEG directives as well as in arranging them to produce the required overlay structure.

The directive may begin in any column following the blank that terminates the label field or, if there is no label field, may begin in any column except column one. It must be terminated by a blank. The Linkage Editor directives are: NAME, ENTRY, SEG, USE, and END.

The operand fields are defined separately for each directive. Blanks within the operand fields are completely ignored by the Linkage Editor; the blanks preceding and following the directive are the only ones considered. Continuation of an operand field is indicated by the use of a semicolon. It may appear at any place in the operand field. Columns 73-80 of the card image are not included in the operand field; they may be used as a sequence field.

NOTE

The MRX Assembler does not accept continuation on SEG statements presented to it embedded in assembly language code.

NAME DIRECTIVE

The NAME directive specifies the beginning of a directive set within a directive file, and designates the following directives as comprising the map of the named program. It must be the first directive in the set. The directive set is terminated by an END directive. The NAME directive may appear only in a directive file and may not be coded within an object program. The content is:

```
label     NAME
```

Label is the name specified as the operand of the PGM keyword on the //PAR statement and designates the basic input module or directive set.

ENTRY DIRECTIVE

The ENTRY directive provides the ability to redefine the Primary Entry Points of any load modules to be generated. This directive may be used to specify a new Primary Entry Point for either an entire program or a sub-complex. The ENTRY directive always appears after the SEG directive that defines the program or sub-complex which it will modify, and before another SEG directive is encountered. The ENTRY directive may be used only in a directive file and may not be coded in an object module.

The form of the ENTRY directive is:

```
label      ENTRY      operand
```

Label is the name of the load module.

Operand has the form: symbol [+'hexadecimal literal']

In the operand, symbol is the name of an entry point or control section presently part of the program. The optional hexadecimal literal is the offset from the named entry point. The hexadecimal value, if present, must be enclosed in apostrophes.

Example:

In this example, the location of the Primary Entry Point for the load module named PROFIT is changed to 2F (hexadecimal) locations beyond the entry point named PROFIT12.

```
PROFIT     ENTRY      PROFIT12+'2F'
```

USE DIRECTIVE

The USE directive is used in defining the Entry Point Search Domain for a given module. It provides the ability to control and direct the Linkage Editor entry point search while resolving the external references of a given load module. The sequential order of the USE directives for a module determines the search hierarchy of the Entry Point Search Domain.

This directive is required only if duplicate entry points exist. This occurs when a request has been made (via the SEG directive) to allocate an object module more than once. The USE directive may be used only within a directive file where any number of USE directives may be used. The USE directive may not be coded in an object module.

The content of the USE directive is:

```
label2     USE      label3
```

Label2 specifies the name of the load module containing the externals to be selectively resolved, as specified on the SEG statement. Label2 must have been previously defined.

Label3 must be the name (label field) of a SEG statement naming the load module whose entry points are to be used while resolving externals declared in the module designated by label2, as specified on a SEG statement. Label3 must have been previously defined. Neither label2 nor label3 can be sub-complex names; they must be names of SEG statements resulting in load modules.

Example:

In the example, the load module named FMOD3 will be searched for entry points to satisfy the external references in load module AMOD before any other modules in the Library Search Domain are searched.

```
AMOD       USE      FMOD3
```

END DIRECTIVE

The END directive terminates a directive set within the file specified by ID=DIR. The END directive must always be the last statement of a directive set and may appear only in a directive file and not within an object module.

The statement has only one field, the directive. There are no label or operand designations. The directive may begin in any column other than column one.

SEG DIRECTIVE

The SEG directive is used to specify additional input to the Linkage Editor as well as the physical organization of input modules and control sections of the output program. The relationships of load modules to be generated in an overlay program and their physical organization is defined via the SEG directive. Previously designated search hierarchies for resolving external references may also be overridden by the SEG directive specifying the library in which the module or control section must be located.

SEG directives may appear anywhere between a NAME directive and an END directive, with one restriction. Any SEG directive which references the label field of another SEG directive as an operand must physically follow the referenced SEG statement. This requirement occurs because the Linkage Editor does not look ahead among the SEG directives for undefined terms. When a module name referenced as an operand is not located as a SEG statement label in a backward search, the Linkage Editor searches the appropriate library or libraries for the module.

Example:

In this example the sub-complex defined by SEG A is referenced in SEG B, and the sub-complex defined by SEG B is referenced in SEG C. The reference to D in SEG C will not bring the sub-complex defined by SEG D, but will cause the module D to be collected. Module D, however, is not prohibited from containing a SEG statement of the same name, in effect causing a forward reference.

Within any given execution of the Linkage Editor, nested sub-complex definition within modules is allowed to 4 levels. SEG statements that exist within a module are counted as one of those 4 levels.

```
A       SEG     M1,M2,M3
B       SEG     G+A
C       SEG     D+B
D       SEG     M4,M5
```

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:

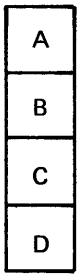
+ plus	Inclusion Operator: Terms separated by a plus sign are considered to be included sequentially in memory in the order encountered. This results in simultaneous memory occupation of the modules named by these terms.
, comma	Exclusion Operator: Terms separated by a comma are considered to occupy 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
```

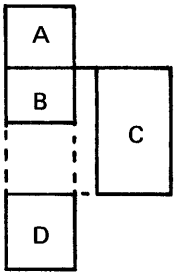


Modules A, B, C, and D will all be loaded together when the load module ALPHA is called. None of the modules can be loaded separately, since only one load module is produced.

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
```

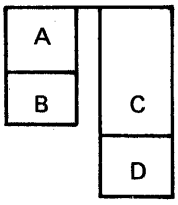


Load module C is link-edited to overlay module B. Module D is link-edited so that it will be loaded with modules A and B, but will occupy space following the area in which load module C will be loaded.

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.

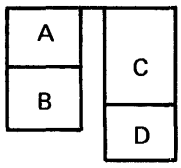
```
GAMMA SEG A+B,(C+D)
```



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.

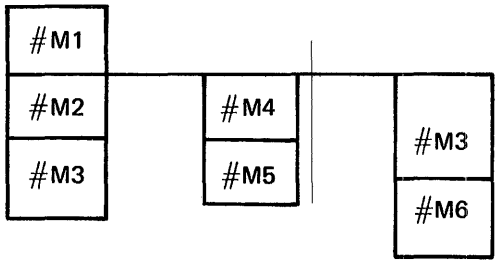
Each CSECT (not declared common) specified in a SEG directive causes storage to be allocated, even though the same CSECT name may be specified more than once in the SEG operand.

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

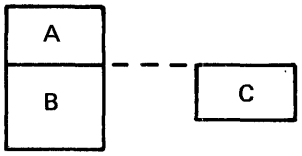
The following are examples of SEG statements used to obtain the memory configurations diagrammed.

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.



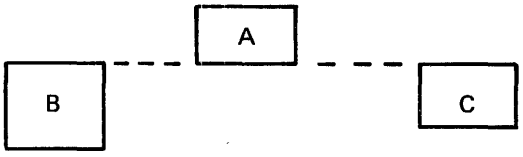
Alternate SEG Statement: [label] SEG A+(B,C)

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)
```

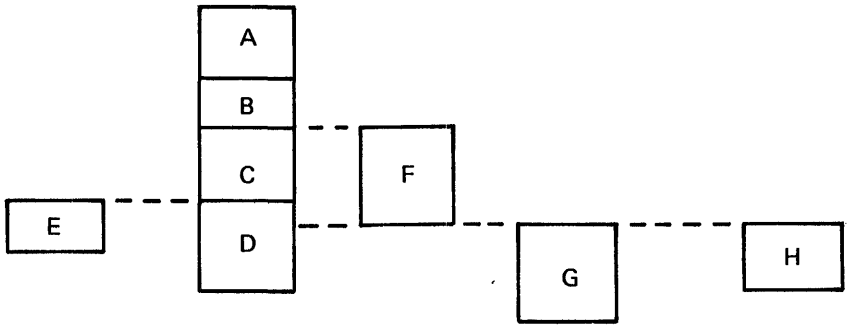
In this alternate example, SEG X defines modules B and C as separate load modules overlaying the same area in memory. Neither B nor C is loaded simultaneously with the root A. The load module shown as B in the illustration will be named X on the library.

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.

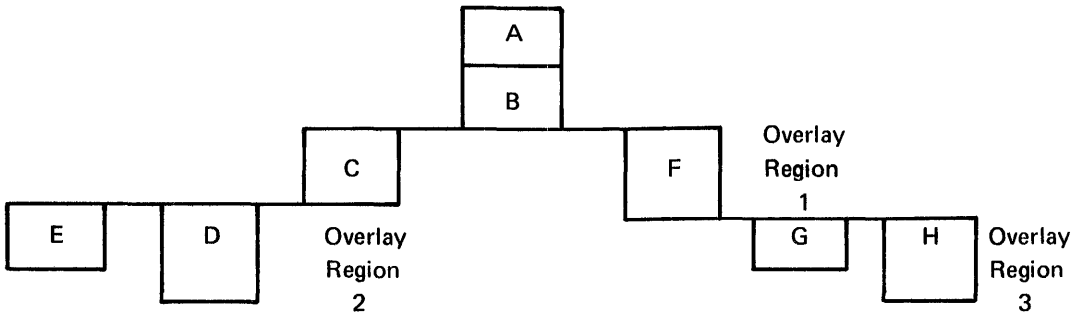
SEG Y defines sub-complex F containing three separate load modules, G, G and H.

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      mmddyy      hhmmss
```

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

mmddyy — Current system date (month, day, year).

hhmmss — 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
```

xxxxxxxx — 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.

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

CM name	A labeled or unlabeled common block.
CS name	A named or unnamed control section.
EP name	A named or unnamed entry point. If duplicate entry points are encountered, the word DUPLICATE appears on the right side of the listing on the same line as the entry point name.
PE name	Primary entry point of the load module.
EX name	An external name.
addr	Relative relocatable address at which the named item appears in the module, except for EX, where it is the relative relocatable address of the entry point in the module satisfying the external. If an external remains unsatisfied, four asterisks will appear in place of an address.
modname	Name of the load module which now contains the entry point satisfying the external (not necessarily the name of the object module in which it was found).

SAMPLE MAP

The map in Figure 3-1 is an example of a link-edit map produced from link-editing an RPG II program. In the example, only SEG statements are shown, since the RPG II compiler automatically produces SEG directives and no further external directive file has been specified. The last SEG statement has no label; therefore the name of the primary load module defaults to the name specified for the PGM parameter (FPT002 in the example).

Not all object modules in the example are shown on the SEG directives; some are included as a result of searching library $SYSOBJLIB for entry points to satisfy externals.

The SEG statements produce modules that occupy main storage space according to the diagram in Figure 3-2.

The sizes shown on the listing in Figure 3-1 for modules $RGRER, $RGRCL, $RGRMR, and $RGRNT are the actual sizes of these modules. The size shown for load module FPT002 is the composite length of the program, including load module FPT002 and the longest overlay, $RGRMR.

Note the unresolved external $WRTBUF in module $DMGPF of load module FPT002. Since the sample map is not intended to reflect an actual program, the unresolved external is included for illustrative purposes only.

```
**LINKAGE EDITOR LEVEL-P   020973   064935
AAAAAAAA SEG ($RGRNT/$SYSOBJLIB),;
   ($RGRMR/$SYSOBJLIB),($RGRCL/$SYSOBJLIB),;
   ($RGRER/$SYSOBJLIB)
   SEG $RGREX/$SYSOBJLIB+AAAAAAAA+FPT002

LOAD MODULE = $RGRER        BSADR=    0B56    SIZE = 0412       ENTRY POINT= 0B56
   $RGRER          0B56
      CM    $RGCOM      0000
      CM    $RGCM2      00EE
      CS                0B56
      EP                0B56                                              DUPLICATE
      EP    $RGCOM      0000                                              DUPLICATE
      EP    $RGCM2      00FE                                              DUPLICATE
      EX    $RGRTN      0176        FPT002

LOAD MODULE = $RGRCL        BSADR=    0B56    SIZE = 046A       ENTRY POINT= 0B56
   $RGRCL          0B56
      CM    $RGCOM      0000
      CM    $RGCM2      00EE
      CS                0B56
      EP                0B56                                              DUPLICATE
      EP    $RGCOM      0000                                              DUPLICATE
      EP    $RGCM2      00EE                                              DUPLICATE
      EX    $RGCL       01F2        FPT002

LOAD MODULE = $RGRMR        BSADR=    0B56    SIZE = 070C       ENTRY POINT= 0B56
   $RGRMR          0B56
      CM    $RGCOM      0000
      CM    $RGCM2      00EE
      CS                0B56
      EP                0B56                                              DUPLICATE
      EP    $RGCOM      0000                                              DUPLICATE
      EP    $RG413      0B78
      EP    $RGORS      0E8E
      EP    $RGOSP      125A
      EX    $RGRT       0156        FPT002
      EX    $RGMI       092C        FPT002

LOAD MODULE = $RGRNT        BSADR=    0B56    SIZE = 053C       ENTRY POINT= 0B56
   $RGRNT          0B56
      CM    $RGCOM      0000
      CM    $RGCM2      00EE
      CS                0B56
      EP    $RGRNT      0B56
      EP    $RG300      0D88
      EP                0B56
      EP    $RGCOM      0000
      EP    $RGCM2      00EE
      EX    $RGORS      0E8E        $RGRMR

LOAD MODULE = FPT002        BSADR=    0000    SIZE = 29AC       ENTRY POINT= 1262
   $RGREX          0000
      CM    $RGCOM      0000
      CM    $RGCM2      00EE
      CS                0140
      EP                0140                                              DUPLICATE
      EP    $RGCOM      0000                                              DUPLICATE
      EP    $RGXLM      091C
      EP    $RGCL       01F2
      EP    $RGRT       0156
      EP    $RGRIN      0176
```

Figure 3-1. Sample Link-Edit Map

```
   EP    $RGEXC    0140
   EP    $RGMI     092C
   EX    $RG300    0D88        $RGRNT
   EX    $DMGPF    20E2        FPT002
   EX    $DMGPR    2252        FPT002
   EX    $DMOCC    23E2        FPT002
   EX    $RG413    0B78        $RGRMR
FPT002       1262
   CS    FPT002    1262
   CS    $RGCOM    0000
   PE    FPT002    1262
   EX    $RGEXC    0140        FPT002
   EX    $RGPRX    2454        FPT002
   EX    $RGOFP    2504        FPT002
$DMGPF       20E2
   CS    $DMGPF    20E2
   EP    $DMGPF    20E2
   EX    $WRTBUF   ****
$DMGPR       2252
   CS    $DMGPR    2252
   EP    $DMGPR    2252
$DMOCC       23E2
   CS    $DMOCC    23E2
   EP    $DMOCC    23E2
   EP    $DMOCC1   23E4
$RGPRX       2454
   CM    $RGCOM    0000
   CS              2454
   EP              2454                                    DUPLICATE
   EP    $RGCOM    0000                                    DUPLICATE
   EP    $RGPRX    2454
$RGOFP       25D4
   CM    $RGCOM    0000
   CS              25D4
   EP              25D4                                    DUPLICATE
   EP    $RGCOM    0000                                    DUPLICATE
   EP    $RGOFP    25D4
```

Figure 3-1. Sample Link-Edit Map (Continued)

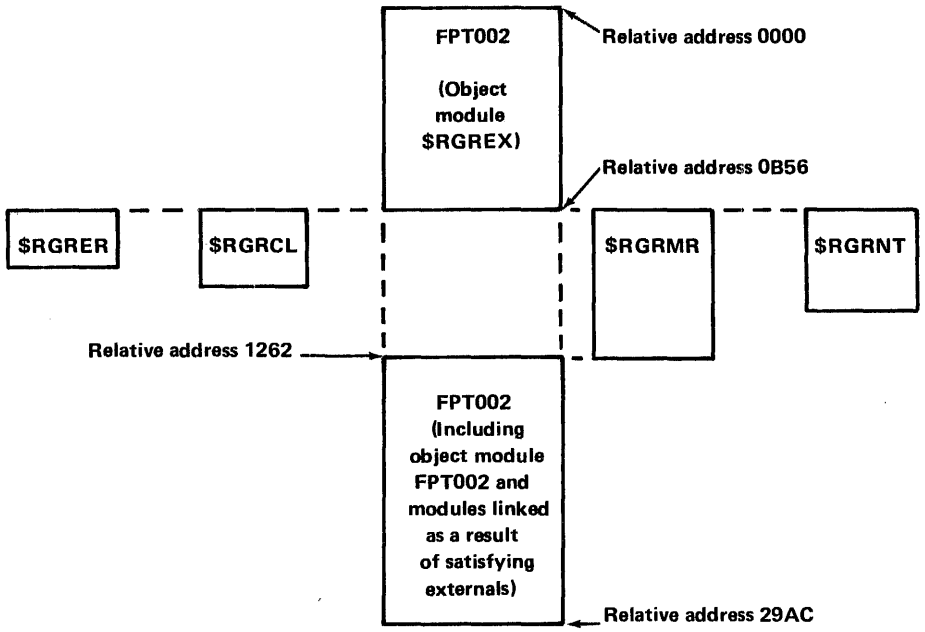
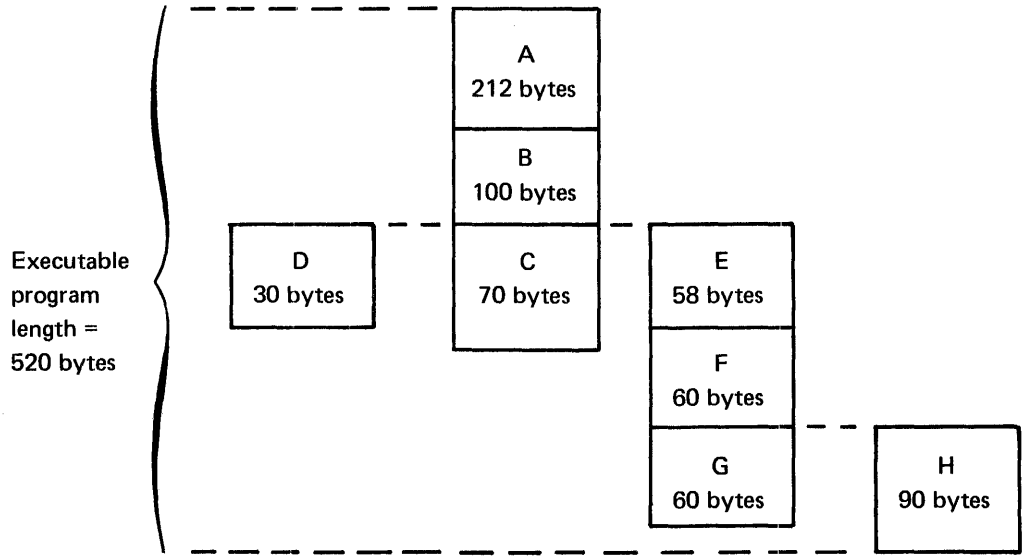


Figure 3-2. Storage Occupation for Sample Program

EXECUTABLE PROGRAM LENGTH

An important programming consideration is the composite length of load modules produced by the Linkage Editor and shown on the link-edit map. The length shown on the map for the root load module consists of the length of the root module plus the length of the longest string of overlay load modules possible in main storage at the same time, if the overlays extend beyond the root module. The length shown for each overlay load module consists of its length plus the length of any logically dependent overlays, if they extend beyond the initial overlay. This can be shown by the following illustration.



In the illustration, the actual size is shown for each segment. However, the composite lengths of the load modules are shown in the following chart. For illustrative purposes, length values are shown in decimal; on the link-edit map they appear in hexadecimal notation.

Load Module	Load Module Size	Composite Length
A (A+B+C)	382 bytes	520 bytes (A+B+E+F+H)
D	30	30 bytes
E (E+F+G)	178	208 bytes (E+F+H)
H	90	90 bytes

The composite length shown for load module A is 520 bytes. The lengths for C and G are not included, since the areas they occupy are overlaid by load modules extending beyond them. The composite length shown for load module E is 208 bytes; H is included because it is linked to overlay the area occupied by G and extends beyond G. The length shown for load module D is 30 bytes and for H is 90 bytes, since neither of these modules have dependent overlays.

Load module A is the primary load module. Therefore, execution of this program requires a partition of at least 520 bytes, plus sufficient space for the partition space pool and partition tables.

JOB STREAM EXAMPLES

The following examples show sample Control Language statements used in three executions of the Linkage Editor.

Example 1: Basic Execution

In this example, an execution of the Linkage Editor is requested. No directive file is specified. The program to be link-edited is member TRANS1 located on library PROJLIB. All control sections and modules required to create the load modules must reside on either PROJLIB or $SYSOBJLIB. The executable load program created will be placed on the system load library, $SYSLODLIB.

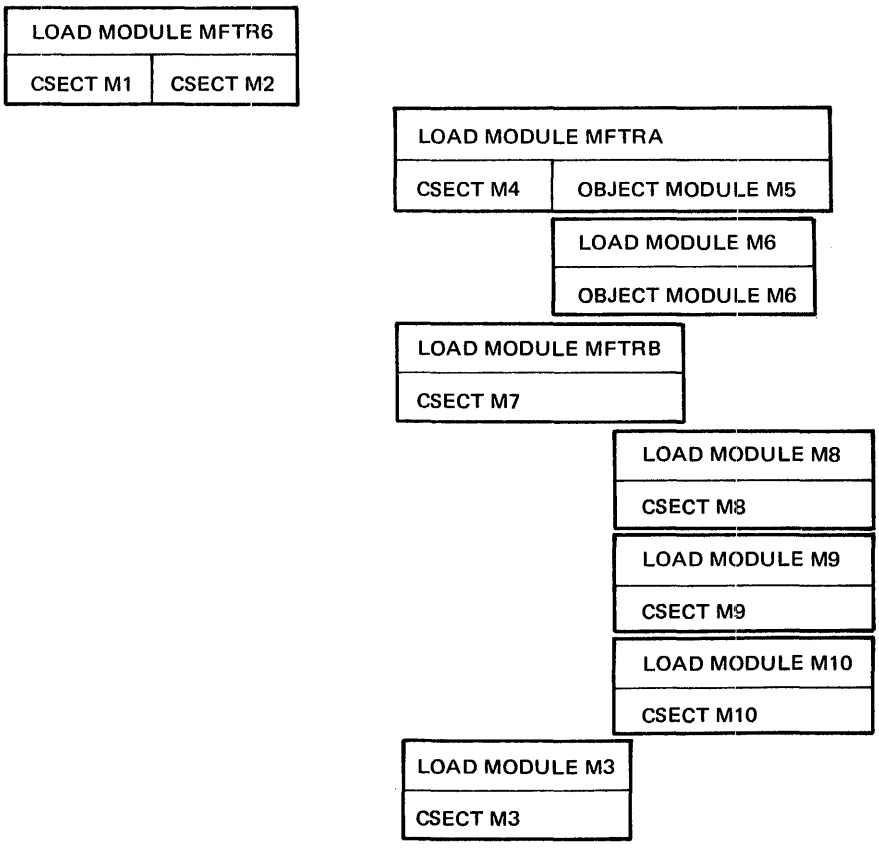
```
//JOB   NAME=SAMPLE
//EX    PGM=LNKEDT
//DEF   ID=LIST,DEV=PRINTER
//DEF   ID=INPUT,FIL=PROJLIB,STA=(P,I)
//DEF   ID=OUTPUT,FIL=$SYSLODLIB,STA=(P,O)
//PAR   PGM=TRANS1
//EOJ
```

Example 2: Basic Execution with a Directive Set

In this example, an execution of the Linkage Editor is requested using a directive file. The program to be link-edited is MFTR6 located on library PROJLIB. All control sections and modules required to create the load modules must reside on either PROJLIB or $SYSOBJLIB. The Linkage Editor directives are located in MFTR2. The executable load program to be produced will reside on PROJLODLIB.

```
//JOB   NAME=SAMPLE2
//EX    PGM=LNKEDT
//DEF   ID=LIST,DEV=PRT
//DEF   ID=INPUT,FIL=PROJLIB,STA=(P,I)
//DEF   ID=OUTPUT,FIL=PROJLODLIB,STA=(P,O)
//DEF   ID=DIR,FIL=MFTR2
//PAR   PGM=MFTR6,LST=XREF
//DATA FIL=MFTR2
MFTR6 NAME
MFTRA SEG  #M4+M5,M6
MFTRB SEG  #M7+(#M8),#M9,#M10
MFTR6 SEG  #M1+#M2+(MFTRA),MFTRB,#M3
MFTR6 ENTRY STRTMAIN
      END
/*
//EOJ
```

The following diagram specifies the modules created for MFTR6, and shows the overlay structure that results.



Example 3: Compile-Link-and-Execute

This example includes a compile step, a link-edit step, and an execute step.

```
//JOB   NAME=SAMPLE3
//EX    PGM=COBOL
//DEF   ID=INPUT,FIL=$SYSSRCLIB,STA=(P,I)
//DEF   ID=OUTPUT,FIL=LNKLIB1,NUM=1000
//DEF   ID=LIST,DEV=PRINTER
//PAR   IMEM=PROG14,OMEM=PROG14,LIST=YES
//EX    PGM=LNKEDT
//DEF   ID=INPUT,FIL=LNKLIB1
//DEF   ID=OUTPUT,FIL=LNKLIB2,STA=T,NUM=1000,BLK=I,SIZE=256,
//DEF   ID=LIST,DEV=PRT
//PAR   PGM=PROG14,XQT=PROG14,LST=XREF,SIZE='4000'
//EX    PGM=PROG14,LIB=LNKLIB2
//EOJ
```

Example 4: Compile-Link-and-Execute Using a Cataloged Procedure Cataloged Procedure

The following cataloged procedure specifies the same COBOL compile step, link-edit step, and execute step as Example 3. The operands which may (or must) be specified on the //CALL statement are designated on the //DECLARE statement of the cataloged procedure. The //DEF card for ID=$LODLIB is required, since LNKLIB2 is uncataloged in this example.

```
//DEC   LANG=COBOL,SRCFIL=$SYSSRCLIB,
//       SRC=EXNAME,LELST=NORM,
//       MAXSIZ=8K,CMPLST=YES
//EX    PGM=&LANG
//DEF   ID=INPUT,FIL=&SRCFIL,STA=(P,I)
//DEF   ID=OUTPUT,FIL=LNKLIB1,NUM=1000
//DEF   ID=LIST,DEV=PRINTER
//PAR   IMEM=&SRC,OMEM=&SRC,LIST=&CMPLST
//EX    PGM=LNKEDT
//DEF   ID=INPUT,FIL=LNKLIB1
//DEF   ID=OUTPUT,FIL=LNKLIB2,STA=(P,O),CAT=NO,NUM=1000,BLK=I,SIZE=256,
//       VOL=VOLA
//DEF   ID=LIST,DEV=PRINTER
//PAR   PGM=&SRC,XQT=&SRC,LST=&LELST
//PAR   SIZE=&MAXSIZ
//EX    PGM=&SRC,LIB=LNKLIB2
//DEF   ID=$LODLIB,FIL=LNKLIB2,STA=(P,O)
```

A call to the above cataloged procedure could appear as follows. The operands on the //CALL statement must include SRC and EXNAME, specified on the //DECLARE as required each time the procedure is requested.

```
//JOB   NAME=SAMPLE4
//CALL  PRO=CMPXQT,SRC=PROG14,EXNAME=PROG14,
//       LINES=2000,LELST=XREF,MAXSIZ=16K
//EOJ
```

4. RELOCATING PROGRAM LOADER

INTRODUCTION

The Relocating Program Loader (hereafter called the Loader) transfers program load modules from the system and private load libraries, on direct access storage, into a program partition of main storage for execution under the Memorex Operating System. The Loader is a system ($NUCLIB resident) module which operates as a System Supervisor Service Program in the system resident area of main storage.

The program load modules transferred into the partition by the Loader are either relocatable or absolute, and are produced by the Linkage Editor from object modules generated by the language processors. In order to be loaded for execution in the program partition, all program load modules must reside on direct access storage libraries.

Modules which reside on other media, such as cards or magnetic tape, must be placed in a library via the LIBUTIL routine.

The Loader is invoked in each of the following circumstances:

- By the System Control Program at system initialization, to load the Input Reader module of Control Language Services.
- By the Step Initiator module of Control Language Services as a direct result of an //EXECUTE statement, to load the program named on that statement.
- By a Service Request from a currently executing program, or load another of its segments or overlays into its assigned area of the program partition.
- By the System Control Program at the end of each step, to load the Step Terminator module of Control Language Services.

Before control is passed to any module of a program, the Loader determines whether Checkout Debugging has been requested. When DEBUG=YES is specified on the //EXECUTE statement for the program, the Loader calls the Debug routine to initialize the module as specified in the Debug Directive File for the program. Then the Loader passes control to the program. (The Debug routine is described in the **MRX/OS Control Program and Data Management Services, Basic Reference** manual. The **MRX/OS Control Language Services, Extended Reference** manual discusses the DEBUG keyword, and describes the Checkout Debugging table and partition space requirements.)

MACRO SPECIFICATIONS

The primary functions of the Relocating Program Loader are available for problem program use via the FETCH and LOAD macros. The following paragraphs describe the format used in coding the macros, and discuss each of the macros and their keyword operands. The format of the Service Request and of the macro expansion for these functions are provided in Appendices E and F respectively.

MACRO FORMAT

In general all macros used in the Memorex Operating System have the same format, as follows:

```
name        operation        keyword-operand(s)
```

The name field is optional. It contains a 1- to 8-character alphanumeric value which specifies the tag for the macro. No embedded blanks are allowed in this or any other field. The name field, when it is used, begins in column 1 of the punched card. It is separated from the following field on the card by one or more blank characters.

The operation field is required and specifies the macro requested. This field is separated from both the preceding and the following fields on the card by one or more blank characters. If the name field is not used, the operation field may begin in column 2 or any following column on the card.

The operand field is required and contains the parameter keywords used with the operation specified in the macro call. Keywords may be listed in any order, separated by commas. Optional parameters are denoted by brackets, [ ]. Parameters with a choice of specifications are denoted by braces, { } . The operand field is separated from the preceding field by one or more blank characters. Addresses (symbolic locations) used as operands conform to the rules for symbolic addresses as described in the **MRX/OS Assembler Reference** manual.

FETCH MACRO

The FETCH 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 always loaded into the partition at the relative load address specified by the Linkage Editor. Control is always transferred to the module being loaded, either at the primary entry point defined at link-edit for FETCH by module name, or at the entry point specified in the macro for FETCH by entry point.

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.

5. LIBRARY OVERHEAD

INTRODUCTION

This section provides formulas for estimating library storage overhead in blocks. Overhead estimates are described for two categories of library member types: *source,* comprising source, macro, and procedure members; and *encoded,* comprising object, relocatable load, and absolute load modules.

SOURCE CATEGORY

The formula for estimating overhead for members in the source category is as follows:

$$OV = I + \frac{C}{ME} + EOF \times C$$

OV Overhead in blocks.

I Index blocks of 434 bytes divided by the block size of the library file, including the common stored data format (CSD) header (if used), rounded up.

C Number of members to be contained in the file.

ME Number of member entries per catalog block, calculated by the formula:

$$\frac{\text{Block size-78}}{34} \text{ (rounded down) + 1}$$

Remaining space at the end of each block less than 70 bytes is not used; the first eight bytes of each catalog block are used for header information.

The value $\frac{C}{ME}$ is rounded up to the next higher number.

EOF Number of end of file blocks per member; members in the source category require only one.

Example:

Assume an allocated block size of 80 bytes, blocked one record per block, in CSD, with 7 members.

$I = \frac{434}{84}$ rounded up = 6 index blocks

$ME = \frac{84-78}{34}$ rounded down + 1 = 1 catalog block per member

EOF = 1 end of file block per member

$OV = 6 + \frac{7}{1} + 1 \times 7 = 20$ blocks library overhead

ENCODED CATEGORY

The formula for estimating overhead for members in the encoded category is the same as for the source category. However, values for the variables are different.

Formula:

$$OV = I + \frac{C}{ME} + EOF \times c$$

I — Index blocks of 434 bytes divided by block size (restricted to 256 bytes) rounded up = 2.

ME — Number of entries per catalog block of 256 bytes, calculated by the formula $\frac{256-78}{70}$ (rounded down) +1 resulting in a value of 3 for estimating purposes. Entries are variable in length up to 70 bytes; remaining space less than 70 bytes is not used; the first eight bytes of each catalog block are used for header information.

C — Number of members to be contained in the file. The value $\frac{C}{ME}$ is rounded up to the next higher number.

EOF = Number of end-of-files per program module per member. OBJ code has at least three data subsets, each requiring one end-of-file block per member:

- Text
- Relocation Group Dictionary
- Entry Point List

OBJ code may also have an optional subset requiring a fourth end-of-file block per member:

- SEG Statements

REL code has three subsets for root modules, each requiring one end-of-file block per member:

- Text
- Relocation Control Stream
- Composite Entry Point List

REL code has two subsets for overlay modules each requiring one end-of-file block per member:

- Text
- Relocation Control Stream

ABS code has two subsets for root modules, each requiring one end-of-file block per member:

- Text
- Composite Entry Point List

ABS code has one subset for overlay modules, requiring one end-of-file block per member:

- Text

Example:

Assume a library containing seven members, all object type with SEG statements.

I = 2 index blocks

$\frac{C}{ME} = \frac{7}{3}$ rounded up = 3 catalog blocks

EOF = 4 per member

Therefore, 2 + 3 + 4 x 7 = 33 blocks estimated library overhead.

ESTIMATING MODULE SIZE

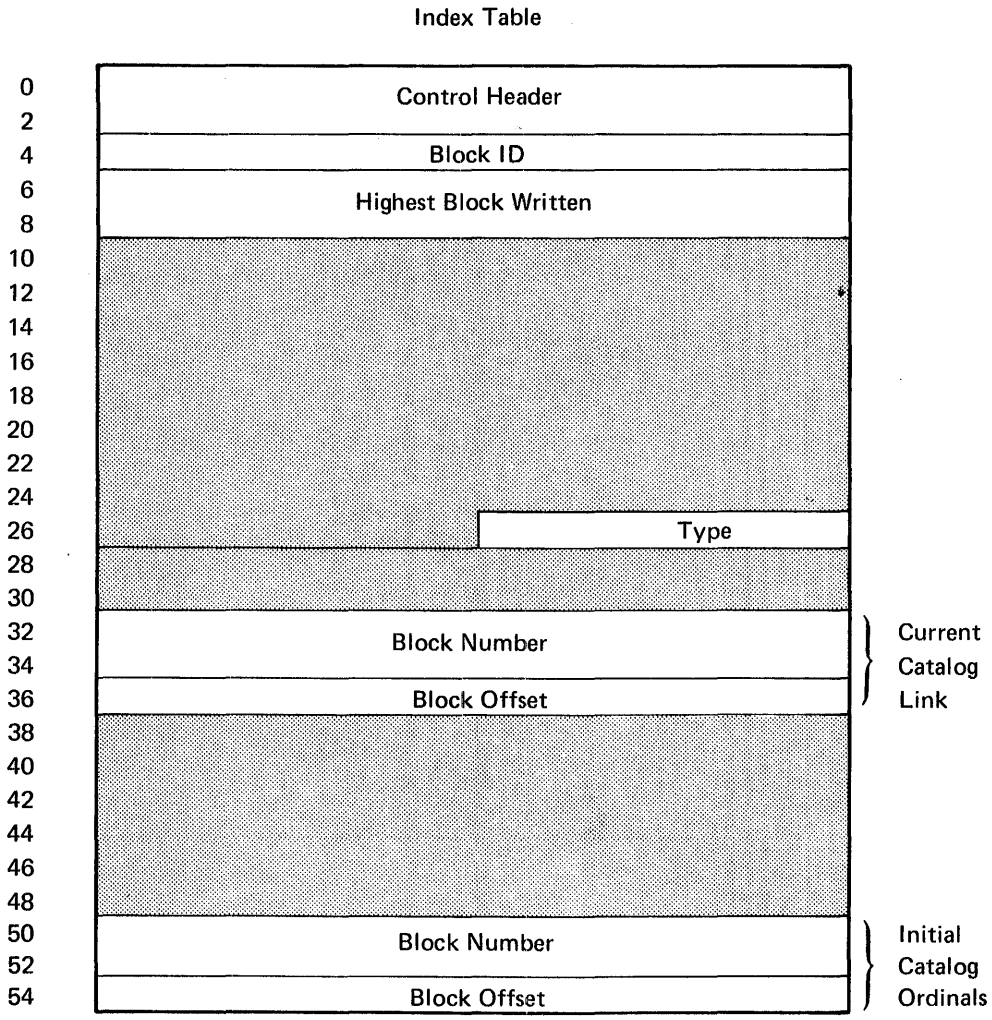
If disc file space is limited, or if the user wants to optimize his use of disc space, a trial compilation or assembly is recommended before a library is permanently allocated.

A. LIBRARY TABLE FORMATS

The following diagrams show the formats of the Index Table, the Catalog Block and the Member Definition Block.

INDEX TABLE

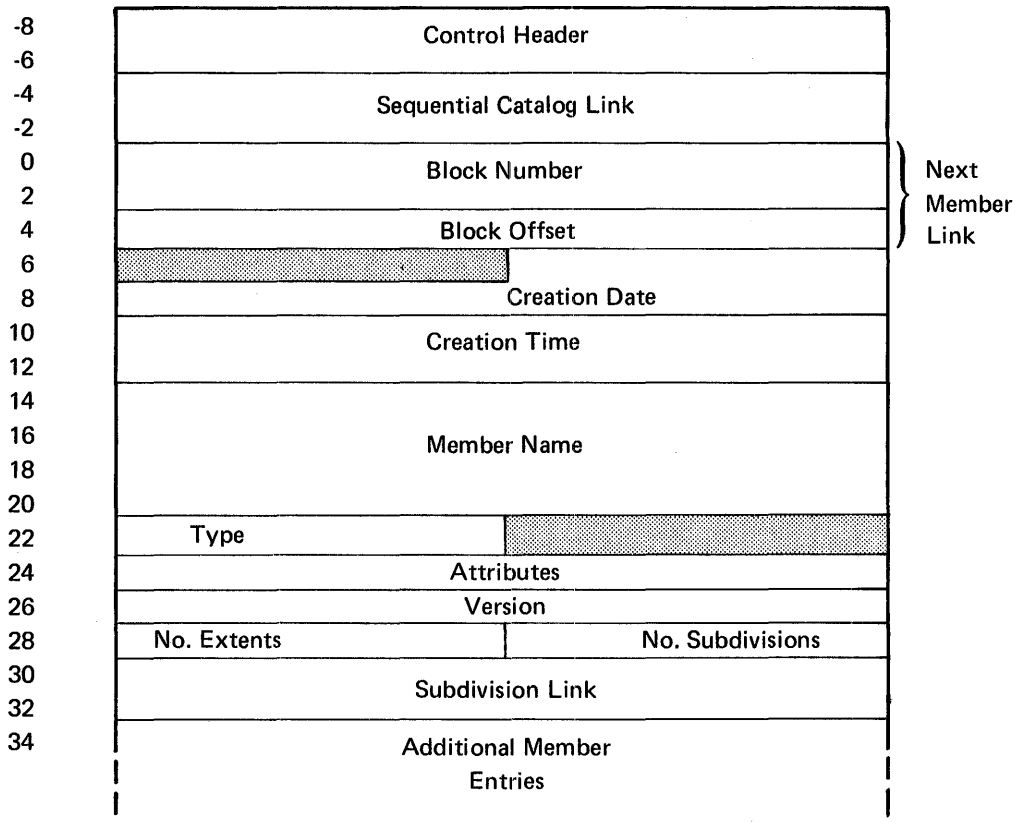
There are 64 catalog ordinals in the index table. The block size for the library determines how many blocks are required for the table. Each additional block contains control header, catalog ordinals, and space headers.



Name	Bytes	Description
Control Header	0-3	Common stored data format standard record header.
Block ID	4-5	Indicator which identifies block as index table of library.
Highest Block Written	6-9	Specifies the highest block written in the library.
	10-26	Reserved for system use.
Type	27	Optional indicator for control of library processing; hexadecimal value specifies type of library. 38 Encoded 86 Symbolic FF All
	28-31	Reserved for system use.
Block Number	32-35	Block number of currently active catalog.
Block Offset	36-37	Byte displacement within the block.
	38-49	Reserved for system use.
Block Number	50-53	Initial block number of the catalog associated with this member chain.
Block Offset	54-55	Byte displacement within the block.

CATALOG BLOCK

Catalog Block

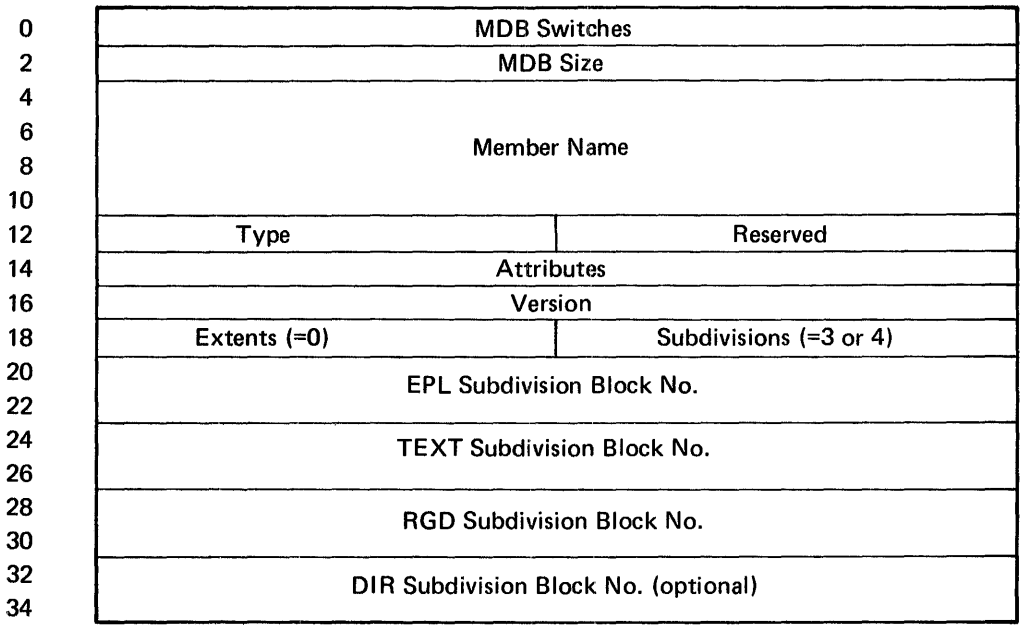


Name	Bytes	Description
Control Header	-8 – -5	Common stored data format standard record header.
Sequential Catalog Link	-4 – -1	Points to next catalog block in this library; last link is zero.
Next Member Link	0-5	
Block Number	0-3	The associated catalog block number which contains the next member entry 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: yy year jjj Julian date

Name	Bytes	Description
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 OBJECT MODULES

Member Definition Block for Object Modules

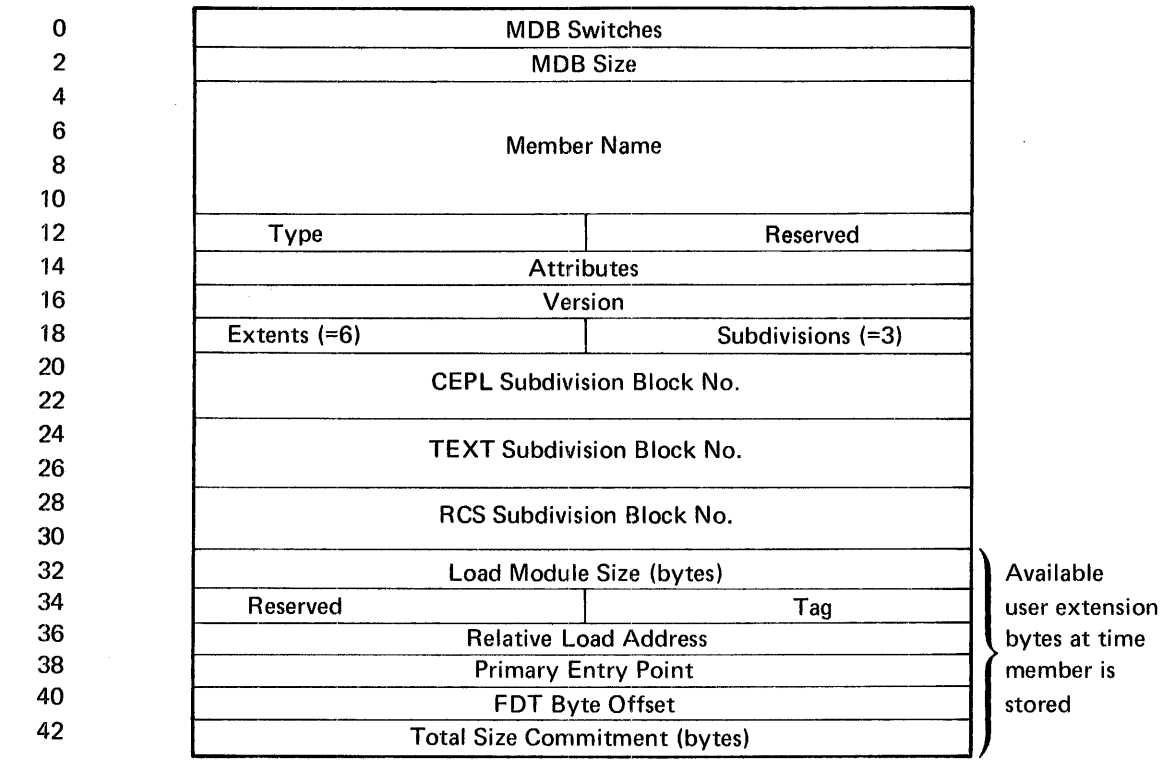


Name	Bytes	Bits	Description
MDB Switches	0	0	Indicates member could not be found in the library search.
	0	1	Member found but store was requested in an ADD mode.
	0	2	I/O error during library function.
	0	3	Reserved for system use.
	0	4	0 indicates REPLACEMENT mode store. 1 indicates ADD mode store.
	0	5	Delete member when found.
	0	6-7	Reserved for system use.
	1	0-7	Reserved for system use.
MDB Size	2-3	0-7	Length of MDB in bytes, not including MDB cells.
Member Name	4-11	0-7	Member identification; 1-8 alphanumeric characters, left justified, blank filled.

Name	Bytes	Bits	Description
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 = 3 subdivision descriptors.
EPL Subdivision Block No.	20-23	0-7	Initial block number of the Entry Point List subdivision.
TEXT Subdivision Block No.	24-27	0-7	Initial block number of the TEXT subdivision.
RGD Subdivision Block No.	28-31	0-7	Initial block number of the Relocation Group Dictionary subdivision.
DIR Subdivision Block No.	32-35	0-7	Initial block number of the optional Directive subdivision.

MEMBER DEFINITION BLOCK (MDB) FOR LOAD MODULES

Member Definition Block for Load Modules



Name	Bytes	Bits	Description
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.

Name	Bytes	Bits	Description
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.

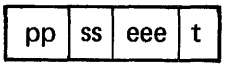
Name	Bytes	Bits	Description
Relative Load Address	36-37	0-7	Relative load address of load module.
Primary Entry Point	38-39	0-7	Relative load address of primary entry point.
FDT Byte Offset	40-41	0-7	Value specified by the POOLSIZ parameter.
Total Size Commitment	42-43	0-7	Composite length of load module.

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:



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: o N = 0 or unspecified which has no meaning in an insert directive, o N = 1 and the input member position is past 1.

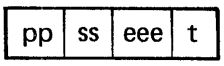
ERROR CODE	EXPLANATION OF ERROR CODE
LBER0418	A sub-parameter length exceeds the maximum.
LBER0428	A right parenthesis is missing from a sub-parameter specification.
LBER0438	A PAR card was not ended by a blank or comma.
LBER0448	A parameter length exceeds the maximum.
LBER0458	A literal string length 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
LBER0648	A type 1 member. MEM=(input-member,member-type) is specified and is illegal. UPDATE needs an output member name.
LBER0658	No member is specified for the UPDATE function.
LBER0668	A library block size exceeds the maximum buffer size.
LBER0678	Null input to update create mode.
LBER0688	Invalid UMODE specification.
LBER0698	N, M of updated directive is not sequential.
LBER0708	N or M of update directive exceeds length of sequence field specified in SEQPOS.
LBER0718	Null input to load function.
LBER0728	The OLIB type parameter is invalid (i.e., is not SYM, ENC or ALL).
LBER0738	The numeric parameter is larger than 5 digits.
LBER0748	The output library block SIZE= parameter is less than the minimum 84 bytes.
LBER0758	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 PAGSIZ.
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:



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.

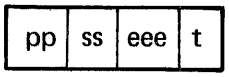
t is a single digit which is always 8 meaning that all the errors are fatal errors.

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	INVALID LIBRARY DEFINITION**** One of the following conditions has been detected: • The file is not sequential. • The library index block is not in the proper format. • The library type does not match the type specified in the library OPEN packet.
***	2F02	LBOP0028	THE FDT FOR THIS LIBRARY COULD NOT BE FOUND IN THE FDT CHAIN**** Either the system failed to open the library or the partition was destroyed.
***	2F03	LBOP0038	INCONSISTENT USAGE SPECIFICATION IN FDT**** One of the following conditions has been detected: • Library was opened with undefined USAGE= keyword specified. • Library has been opened for I/O. • Library has been opened for input and HBW=0.
***	2F04	LBTR0048	I/O ERROR ON LIBRARY**** A disc I/O error has occurred on a library.
***	2F05	LBST0058	END OF ALLOCATION REACHED DURING ATTEMPTED STORING OF A MEMBER ENTRY**** End of disc allocation has occurred for this library.
***	2F06	LBSE0068	MEMBER TYPE, FOUND IN MDB FOR SEARCH OR STORE, IS INVALID FOR THE SUBJECT LIBRARY**** 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.

C. LINKAGE EDITOR OBJECT-TIME ERROR MESSAGES

The following Linkage Editor messages are printed at object time on the printer output printer listing. Each message text is preceded by an 8-character error code that has the following format:



Where: pp is always LE meaning Linkage Editor.

ss is always Ps where P signifies pass and s is the pass number (1-4) that is supplied when the message is printed.

eee is a 3-digit error number specifying the error within the Linkage Editor.

t is a 1-digit number specifying the type of error where:

0 = nonfatal, information message
8 = fatal message

The xxxxxxxx in the message text is replaced by a 1-8 character label or name when the message is printed.

ERROR CODE	MESSAGE TEXT
LEPS0008	VIRTUAL TABLE ACCESS. Any I/O error on disc file used for internal storage (IDENT=VTFLE).
LEPS0018	PARAMETER ERROR. Any invalid keyword or keyword operand submitted on //PAR card.
LEPS0028	PROGRAM EXCEEDS 65K.
LEPS0038	INVALID SEGMENTATION xxxxxxxx. Logical contradiction in structure defined by SEG cards.
LEPS0048	OBJECT TEXT ERROR xxxxxxxx. Bad output from compiler or language processor. (Run job step again with DUMP=YES.)
LEPS0058	UNDEFINED TERM xxxxxxxx. Label or operand of an ENTRY/USE directive is undefined.
LEPS0068	INVALID INPUT MODULE xxxxxxxx. Improper use of relocatable load modules.

ERROR CODE	MESSAGE TEXT
LEPS0078	SYNTAX ERROR. Improper syntax in operands of directives.
LEPS0088	COMMON ALLOCATION ERROR xxxxxxxx. An attempt was made to expand labeled common after it was allocated.
LEPS0108	EXPRESSION STACK OVERFLOW. Currently active SEG terms > 80.
LEPS0118	UNABLE TO LOCATE xxxxxxxx. Unable to find module or control section specified in a SEG term or unable to find directive set specified in PGM= parameter.
LEPS0128	INVALID CSECT ORIGIN xxxxxxxx. Attempted to preset labeled common at an address other than the one already allocated.
LEPS0138	MISSING CSECT xxxxxxxx. Missing control section.
LEPS0148	DUPLICATE TERM xxxxxxxx. The duplicate term name is xxxxxxxx.
LEPS0158	SEGMENTATION EXCEEDS 04 LEVELS. More than 4 concurrently active (nested) object modules contain SEG directives.
LEPS0168	UNABLE TO CATALOG xxxxxxxx. Error returned from LIBSTR; probably end of allocation or other I/O error.
LEPS20170	PROGRAM EXCEEDS SIZE. Primary load module size exceeds the SIZE parameter.
LEP10180	UNDEFINED LIBRARY. Library name specified in SEG term was not included in LSD= keyword. Library will default to INPUT library.
LEP20190	NULL LOAD MODULE. User built load module containing no object text. CSECT's were allocated and entry points established but the load module is not cataloged. User cannot attempt load of "dummy" load module.

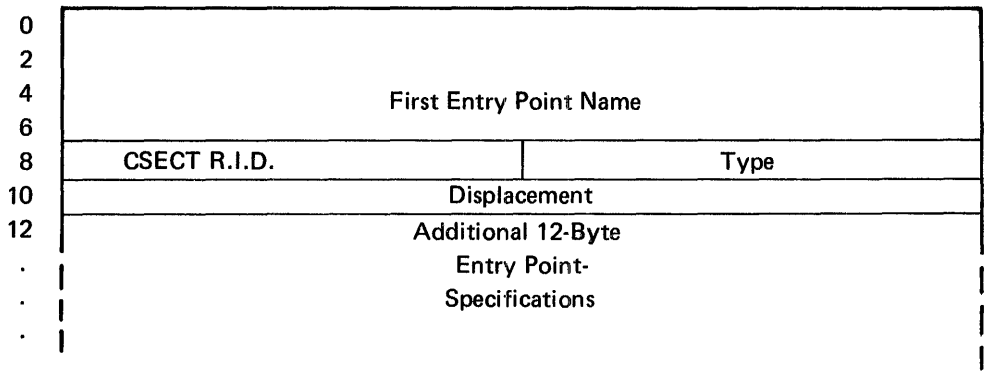
D. OBJECT MODULE STRUCTURE

The following paragraphs and diagrams show the structure of the Entry Point List, Text String Group, and Relocation Group Dictionary in the object module.

ENTRY POINT LIST

The Entry Point List consists of one twelve-byte entry for each entry point defined in the module. The Entry Point List is physically the first block of the object module.

Entry Point List



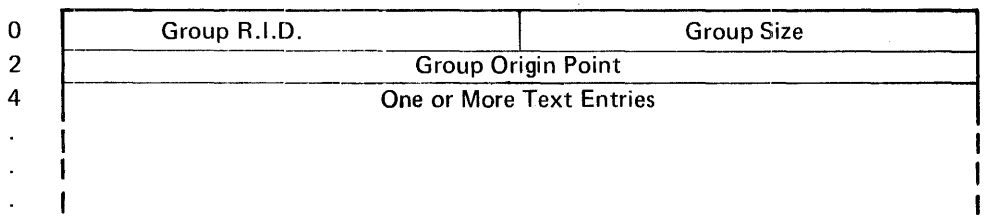
Name	Bytes	Description
Entry Point Name	0-6	Eight byte EBCDIC character string.
CSECT R.I.D.	8	Relocation Identifier (ordinal) of the 'CSECT' item (in the Relocation Group Dictionary) under which this entry point occurs.
Type	9	Value indicating type of name: 1 = Basic entry point 4 = Primary entry point
Displacement	10-11	Offset in bytes from start of 'CSECT' to this entry point.

TEXT STRING GROUP

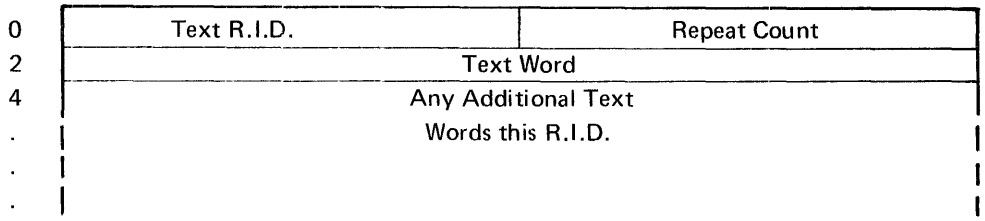
Text String Groups are strings of instructions and data which form the actual program. Text String Groups are grouped according to Control Section with each byte/word accompanied by a relocation attribute code. If the byte pair is to be modified, the Relocation Identifier (attribute code) specifies which 'CSECT', 'COMMON', or 'EXTERNAL' is to be used for the modification.

A Text String Group is composed of a Group Header and one or more Text Entries, formats specified in the following:

Group Header



Text Entry



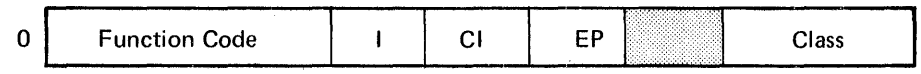
Name	Bytes	Description
Group R.I.D.	0 (of Group Header)	Ordinal in Relocation Group Dictionary of the Control Section (CSECT) under which following Text is generated, value of which is used to modify Group Origin Point.
Group Size	1 (of Group Header)	Total number of Text bytes, this group.
Group Origin Point	2-3 (of Group Header)	Current displacement from beginning of CSECT to be used as relative starting address for following Text items.

Name	Bytes	Description
Text R.I.D.	0 (of Text Entry)	Ordinal in Relocation Group Dictionary of CSECT, COMMON, or EXTERNAL symbol value by which the following Text word(s)/byte(s) must be modified. Values: 1-252_{10} (1-FC_{16}) Identify a CSECT, COMMON, or EXTERNAL; applied to a full word. 0 = Number in Repeat Count; specifies absolute words. 255_{10} (FF_{16}) = Number in Repeat Count specifies absolute bytes. 254_{10} (FE_{16}) Following absolute byte; left adjusted in word; to be propagated as indicated in Repeat Count. 253_{10} (FD_{16}) Following bytes indicated by Repeat Count are segment tags to be modified by the Relocating Program; loader to reflect actual memory allocation provided by the system (not generated by any compilers for this release).
Repeat Count	1 (of Text Entry)	Count, less one, of immediately following Text words/bytes which share the relocation attribute specified by TEXT R.I.D.
Text Word	2-3 (of Text Entry)	For Text R.I.D. = 1-252_{10}, a 16-bit unsigned integer intended as a positive offset from the value derived from Text R.I.D. For Text R.I.D. = 0, 253-255_{10}, see Text R.I.D. for description.

Following is a sample block showing two Text String Groups.

E. SERVICE REQUEST EXPANSION

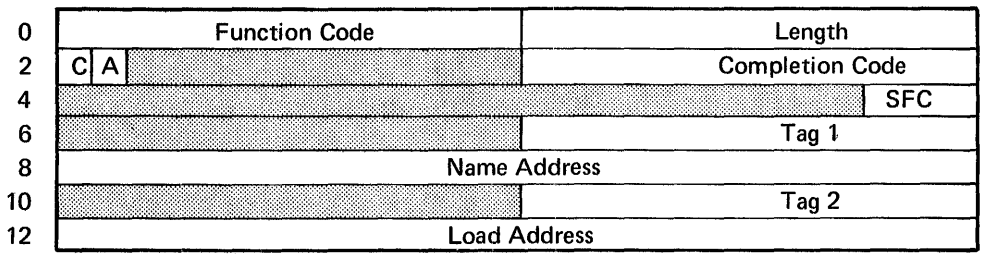
Service Request



Name	Bytes	Bits	Description
Function	0	0-7	Function code mnemonic for all Service Requests is SR.
I	1	0	Indicates the location of the parameter string (LIST keyword): 0 = immediately following the SR 1 = at the address contained in R6
CI	1	1	Indicates when control is to be returned to the requester (RETURN keyword, LOAD macro): 0 = after the SR (LOAD) is completed 1 = after the SR is recognized by Control Program
EP	1	2	Indicates if the requester will process exception completion of the request (ERRCOMP keyword): 0 = requester will not process exception completion (will be processed by Step Terminator) 1 = requester will process exception completion
	1	3	Reserved for system use.
Class	1	4-7	Denotes major class in which the SR falls: 0 = Debugging request 1 = Restricted Service Request issued only by systems programs 2 = Request for Control Program service 3 = BIO Service Request 4 = PIO Service Request 5 = Supervisor Service Request, providing functions such as Data Management and Library Services 6 = TCOM Service Request 7 = UNIT PIO Service Request

F. MACRO EXPANSION FOR LOADER SERVICE REQUEST

Loader Service Request



Name	Bytes	Bits	Description
FC	0	0-7	Function code = 10_{16}.
Length	1	0-7	Length of macro expansion packet including this word: 7 words if SFC=1 or SFC=3; otherwise 5 words.
C	2	0	Request complete indicator.
A	2	1	Abnormal completion indicator.
	2	2-7	Reserved for system use.
Completion Code	3	0-7	Completion code: 01 I/O error 02 Cannot locate entry point 03 Cannot locate module 04 Program exceeds partition limits 05 Invalid load address
	4	0-7	Reserved for system use.
	5	0-4	Reserved for system use.
SFC	5	5-7	Subfunction code: 0 LOAD by entry point 1 LOAD by entry point at load address 2 LOAD by module name 3 LOAD by module name at load address 4 FETCH by entry point 5 Illegal 6 FETCH by module name 7 JMSI FETCH request (Reserved for use by Control Language Services)

Name	Bytes	Bits	Description
	6	0-7	Reserved for system use.
Tag 1	7	0-7	Reserved to provide for upwards compatibility to machine enhancements, particularly the Hardware Relocation Feature, without program modification.
Name Address	8,9	0-7	Address pointer to the 8-byte module name or entry point name.
	10	0-7	Reserved for system use; occurs only if SFC=1 or SFC=3.
Tag 2	11	0-7	Reserved to provide for upwards compatibility to machine enhancements, particularly the Hardware Relocation Feature, without program modification; occurs only if SFC=1 or SFC=3.
Load Address	12,13	0-7	Relative load address; occurs only if SFC=1 or SFC=3.

G. LIBRARY MACROS

The following library macros operate in the System Resident area and are used exclusively by the system to access and manipulate libraries. These macros are not supported for use in problem programs.

LIBOPEN	Opens libraries for the system.
LIBSRCH	Provides the location of a specific member in a partitioned data file.
LIBSTOR	Requests storage or modification of information in the library directory.
LIBCLOSE	Provides for the creation or modification of a catalog entry in the directory of the partitioned data file.

The following buffered library generator macros are supplied only as in-line code to the Linkage Editor, the Relocating Program Loader, and the language processors and translators (Assembler, COBOL, FORTRAN, and RPG II). These macros are not available for use in problem programs.

LGOPENSEG	Opens a segment.
LBCLOSEG	Closes a segment.
LGPUTITM	Writes a variable length item.
LGSTRING	Starts a text string for a language processor.
LGPUTTXT	Inserts items in the text string group for a language processor.

H. SYSTEM LOADER ERROR MESSAGES

The System Loader issues only SYSOUT file error messages. Each message is preceded by 3 asterisks, a 4-digit hexadecimal system-oriented code, and an 8-character error code that has the following format:

pp	ss	eee	t

Where: pp is always OS meaning operating system.

ss is always LD meaning Loader subprogram error.

eee is a 3-digit error number specifying the error within the Loader program.

t is a 1-digit number specifying the type of error. Loader errors are all type 8, which is fatal.

The message text follows the error code. The text of the message ends with 4 asterisks.

MESSAGE ID	ERROR CODE	MESSAGE TEXT
***	OSLD0018	DISC ERROR OR UNEXPECTED EOF (LOADER) ****
***	OSLD0028	UNABLE TO FIND ENTRY-POINT-NAME (LOADER) ****
***	OSLD0038	UNABLE TO FIND MODULE-NAME (LOADER) ****
***	OSLD0048	PROGRAM EXCEEDS PARTITION SIZE (LOADER) ****
***	OSLD0058	OVERLAY EXCEEDS PARTITION LIMITS (LOADER) **** Either overlay is too big or start address is not within limits.

INDEX

COMMENTS FORM

MRX/OS Program Library Services Reference Manual (2200.005)

Please send us your comments, to help us produce better publications. Use the space below to qualify your responses to the following questions, if you wish, or to comment on other aspects of the publication. Please use specific page and paragraph/line references where appropriate. All comments become the property of the Memorex Corporation.

	Yes	No
• Is the material:		
Easy to understand?	☐	☐
Conveniently organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐
Adequately indexed?	☐	☐

- For what purpose did you use this publication? (reference, general interest, etc.)

  _____

- Please state your department's function: _____

  _____

- Please check specific criticism(s), give page number(s), and explain below:

  ☐ Clarification on page(s) _____

  ☐ Addition on page(s) _____

  ☐ Deletion on page(s) _____

  ☐ Error on page(s) _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

MEMOREX

First Class

Permit No. 14831
Minneapolis,
Minnesota 55427

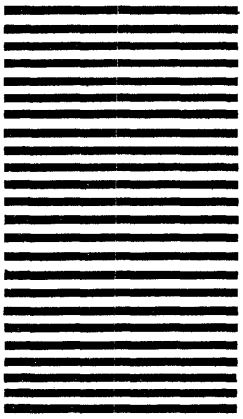
Business Reply Mail

No Postage Necessary if Mailed in the United States

Postage Will Be Paid By

Memorex Corporation

Midwest Operations – Publications
8941 Tenth Avenue North
Minneapolis, Minnesota 55427



. . . . . .

. . . . . .

Thank you for your information. . . . . .

Our goal is to provide better, more useful manuals, and your comments will help us to do so.

. . . . . .Memorex Publications