

---

# Contents

<b>Preface</b>	<b>1</b>
Brief product description	1
Target group	2
Summary of contents	2
Changes since the last version of the manual	4
<b>Introduction to LMS</b>	<b>5</b>
<b>Definitions and conventions</b>	<b>11</b>
What is a library?	11
What libraries are there?	13
Program libraries (PL)	13
Type-related libraries	18
Sequential libraries (archive libraries)	20
What does a program library contain?	22
Member type definition	22
Member designations	25
What do type-related libraries contain?	28
Member type definition	28
Member designation	29
Multiple selection of member designations	31
Construction specification for member designations	34
<b>LMS functions</b>	<b>37</b>
Library assignment	38
Library assignment via LIB	38
Processing of members	42
Adding members to a library	43
Outputting members	46
Listing members	46
Deleting members	47
Numbering records using record numbers and check fields	47
Comparing members	50
Correcting members	53
Renaming members	55
Outputting library directory	55
Storing and calling procedures	56

## Contents

---

Filing members using the delta method	58
Delta as a storage form and organizational aid	59
Adding delta members	60
Overview of delta members	61
Deleting delta members	61
Locking delta members	62
Restrictions when using the delta method	62
Controlling the LMS run	63
Effect of processing operands	63
Controlling log output	66
Control of statement input	68
Controlling screen overflow	68
Execution in run or test mode	70
User interfaces	70
Interrupting the LMS run	71
Using job switches	74
PAM key elimination	75
– Library files	75
– Member processing	76
– Summary	80
<b>Statements</b>	<b>81</b>
Overview of statements	85
ADD      Add data to a library	91
COM      Compare members	105
COR      Correct text members	108
Description of the individual correction statements	112
– *INSERT    Insert records	112
– *DELETE   Delete records	113
– *REPLACE  Replace records	113
– *CHANGE   Change records	114
– *END      Terminate corrections	116
CTL      Define statement input source	117
DEL      Delete members	118
DUP      Duplicate members and duplicate with structure	120
EDT/EDR	
Create, correct and view text members and files	126
END      Terminate LMS run	133
LIB      Assign and close libraries	134
LST      List members	140
NAM      Rename members	142
NOP      Dummy function	144
NUM      Number member records	145
PAR      Set processing operands	147

---

PRT	Control log output	148
RST	Restart after test mode	150
SEL	Output library members to files and FMS libraries	151
SUM	Store comparison statistics	158
SUMPRT	Output comparison statistics	159
SUMADD	Add comparison statistics	159
SUMDEL	Delete comparison statistics	160
SYS	Issue system commands	160
TCH	Change terminal characteristics	161
TOC	Output library directory	162
UPD	Correct object and load modules and LLMs	164
	Description of the correction statements for object modules	168
– *BAS	Define base address	168
– *CON	Define cross control number	168
– *COR	Correct text records	169
– *DEL	Delete parts of object modules	171
– *END	Terminate correction input	172
– *ID	Define identification	172
– *INS	Insert INCLUDE record	173
– *INV	Convert corrections	174
– *NAM	Rename symbols	175
– *REM	Cancel corrections	175
– *REP	Insert REP record	176
– *SET	Modify control section attributes	178
	Description of the correction statements for load modules	185
– *BAS	Define base address	185
– *CON	Define cross control number	185
– *COR	Correct text records	186
– *DEL	Delete correction journal records	188
– *END	Terminate correction input	188
– *ID	Define identification	188
– *REM	Cancel corrections	189
– *SEG	Define segment	189
	Description of the correction statements for LLMs	192
– *COR	Correct text records	192
– *DEL	Delete correction journal records	194
– *END	Terminate correction input	194
– *ID	Define identification	195
– *REM	Cancel corrections	195
USE	Branch to user programs	196
\$	Output statement buffer	201

<b>Processing operands</b>	<b>203</b>
Table of processing operands	205
PAR BASE Define base address	209
PAR CHECK Define check field in input records	210
PAR CSECT Specify a CSECT name	211
PAR COMPARE Control compare function	212
PAR DESTROY Control physical deletion	215
PAR ERRCONS Output messages to SYSOUT	216
PAR FCBTYPE Define FCB type of output file	217
PAR FORMAT Define record format	219
PAR INFO Define scope of output	221
PAR KEY Transfer file attributes and ISAM key	224
PAR LCASE Lowercase/uppercase conversion	225
PAR LINE Define number of lines and columns per log page	227
PAR LOG Log statements	228
PAR LST Define scope and mode of member listings	229
PAR NEWFORM Control form feed	234
PAR OVERWRITE Overwrite identically named members	235
PAR PATH Specify a pathname	236
PAR PHASE Define phase format	237
PAR RANGE Define check field in output records	238
PAR REFERENCE Define reference conditions	239
PAR SEGMENT Define segments of load module	240
PAR SLICE Specify slice	241
PAR SORT Sort directory	242
PAR STRING Define string in check field of output records	243
PAR STRIP Suppress records	244
PAR SUM Generate comparison statistics	245
PAR TERMINATE Control termination procedure in error situations	246
PAR TEST Activate/deactivate and terminate test mode	248
PAR TOC Control output format for directories of program libraries	249
PAR TYPE Predefine member type	251
PAR VALUE Control numbering in check field of output records	252
<b>Examples</b>	<b>253</b>
Simple examples	253
Add, correct and assemble library source programs	253
Duplicate members	258
Compare members	262
Processing delta members	266
Complex examples	269
Correct a source program using COR	269
Correct an object module using UPD	272
Compare members and prepare correction statements	274

Output a member to a file	277
Output comparison statistics	282
Branch to a user program while a member is being listed	285
<b>Old LMS subroutine interface</b>	<b>289</b>
<b>Messages</b>	<b>293</b>
List of messages	293
System queries	317
Access method messages	317
<b>Appendix</b>	<b>325</b>
Conversion of MLU, LMR, COBLUR to LMS	325
BS1000-BS2000 compatibility	326
Statements and processing operands	328
LIBIN Assign input library	329
LIBOUT Assign output library	330
Processing operands	332
– PAR DECOMPRESSED	
Control compression for macros and source programs	332
<b>References</b>	<b>333</b>
<b>Index</b>	<b>339</b>



## Preface

This manual describes the functions and mode of operation of the Library Maintenance System (LMS).

### Brief product description

The Library Maintenance System (LMS) creates and manages program libraries and processes the members they contain.

Program libraries are BS2000 PAM files which are processed using the library access method PLAM (Program Library Access Method); hence they are also known as PLAM libraries.

The main advantages of this method are that

- all member types in a library can be processed via uniform statements,
- members with identical names but differing type/version designations may exist,
- concurrent read/write access to the library by different users is supported,
- standardized data management with unified access functions becomes possible for most of the data elements (=members) created during the software development cycle, and
- the utility routines and compilers can access this data repository and process the individual members directly.

This eliminates many of the problems arising during the creation, maintenance and documentation of programs.

## Target group

This manual is aimed at all BS2000 users who employ libraries to manage their data.

Users should be familiar with BS2000, in particular with its major commands. Appropriate information can be found in the manual "Introductory Guide for System Users" [6].

LMS comes with a separate Ready Reference [13]. This is intended as a guide, i.e. brief operating instructions for the experienced user of LMS. It lists all the LMS statements and processing operands.

## Summary of contents

### Manual division

As of Version LMS V2.0A, the manual has been divided into two volumes for the first time, namely this manual and the "LMS Subroutine Interface" manual. Apart from the ever increasing functionality of LMS and the resulting increasing scope of the manual, the main reason for dividing up the previous manual was the independence of the subroutine from the LMS user interface.

### Manual structure

This manual covers the following topics:

- **Definitions and conventions**  
The member types and library formats that can be processed using LMS
- **LMS functions**  
An outline of the facilities provided by LMS
- **Statements**  
All statements in alphabetical order
- **Processing operands**  
All operands in alphabetical order
- **Examples**  
Selected examples of LMS applications
- **Old LMS subroutine interface**  
Description of the conventions and an example
- **Messages**  
The messages issued by LMS arranged according to their code numbers.



The alphabetical index at the end of the manual permits explanations of key terms and concepts to be located quickly within the body of the text.

Throughout the text, reference literature is quoted using abbreviated titles accompanied by a number in square brackets. The full title of each publication referred to may be found under the appropriate number in the "References" section.



### Changes since the last version of the manual

The following new features have been added since publication of the previous version of the LMS manual (LMS V1.4A):

- What was formerly chapter 8 "LMS as a subroutine" is now described in a separate manual [15].
- The section on "PAM key elimination" (page 75 ff.) has been expanded.
- The new member type F for IFG format masks and member type U for IFG user profiles have been introduced. F and U types can be used in conjunction with the DEL, DUP, LST, NAM and TOC statements.
- Member type L for link and load modules (LLMs) has been newly introduced. L can be used with the DEL, DUP, LST, NAM, TOC and UPD statements.
- The following new processing operands are available for member type L: PAR CSECT, PAR PATH and PAR SLICE.
- The correction statements for object modules have been extended by the correction statement \*REM (page 175).
- The chapter "LMS dialog interface" has been renamed "Old LMS subroutine interface" (page 253).
- The BS2000 commands have been converted from ISP to SDF format.

## Introduction to LMS

LMS creates and manages program libraries and processes the members contained therein. A program library is a file with a substructure. It contains members (also called elements) and a directory (table of contents) listing all the members.

A member is a logically coherent data set such as a file, a procedure, an object module or a source program. Each member can be individually addressed within the library.

Each library has an entry in the system catalog. The user can define its name and other file attributes such as the retention period or shareability.

Storing several files in one library relieves the load on the system catalog, since the latter only contains an entry for the library and no entries for the various members. Storage space is also saved, as the members are stored in the library in compressed form.

If the delta method is used, only the differences (deltas) to each preceding version are stored when a member has several versions. This yields further savings of storage space. When such versions are read, LMS merges these deltas at the appropriate locations so that the complete member is available to the user.

Object modules and load modules can be directly stored in program libraries by the compilers and TSOSLNK respectively. LMS is also capable of copying object modules from the EAM area and load modules from files to the program library.

LMS performs the following functions:

- create libraries
- add members to a library
- edit members
- output members to files
- copy members to another library
- list members
- delete members
- correct members
- rename members
- renumber members
- compare members
- extract and store differences (deltas) between member versions
- output library directory (table of contents)

LMS processes the following library formats:

- **Program libraries** for storing source programs, macros, object modules, load modules, listings, procedures, text, etc.  
These libraries are processed by means of the program library access method (PLAM).
- **Sequential libraries** for storing source programs, object modules, macros and BS1000 phases on tape.

Moreover, LMS can also process any existing MLU, LMR and COBLUR libraries.

Conversion of previous procedures to PLAM is thus considerably simplified and facilitated.

The LMS subroutine interface [15] offers the user convenient options for processing LMS libraries and their contents, direct from the main program, with LMS being loaded dynamically. This subroutine interface can also be used in the XS (extended system) area.

The following figure illustrates the LMS input and output capabilities.



Fig. 1 LMS access options

*Example of an LMS run*

```

/SHOW-FILE-ATTRIBUTES A.
00000003 :N:$USER.A.BEISPIEL
00000003 :N:$USER.A.QUELL.A
:N: PUBLIC: 2 FILES RES= 6 FREE= 3 REL= 0 PAGES
/START-PROGRAM $LMS ----- (01)
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$LIB FILE=UEB.BIBL,BOTH,NEW ----- (02)
$ADDS A.QUELL.A ----- (03)
$PAR LOG=MED ----- (04)
$ADD A.BEISPIEL>BSP ----- (05)
INPUT FILE }
OUTPUT LIBRARY= :N:$USER.UEB.BIBL,DEV=DISK } ----- (06)
      ADD A.BEISPIEL AS (D)BSP/@(0001)/1991-07-24 }
$TOC* * ----- (07)
INPUT LIBRARY= :N:$USER.UEB.BIBL,DEV=DISK }
TYP NAME VER (VAR#) DATE }
(D) BSP @ (0001) 1991-07-24 }
  1 (D)-ELEMENT(S) IN THIS TABLE OF CONTENTS } ----- (08)
TYP NAME VER (VAR#) DATE }
(S) A.QUELL.A @ (0001) 1991-07-24 }
  1 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS }
$END ----- (09)
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/

```

- (01) LMS is called.
- (02) LMS creates UEB.BIBL as a new program library and assigns it as the input and output (I/O) library.
- (03) File A.QUELL.A is added to the library as a member of type S with member name A.QUELL.A.
- (04) Processing operand LOG=MED causes LMS to output not only error messages but also positive acknowledgments.
- (05) File A.BEISPIEL is added to the library as a member of type D with member name BSP.
- (06) Positive acknowledgment: since processing operand LOG=MED has been specified, LMS confirms the addition of file A.BEISPIEL as member BSP.
- (07) LMS is to output the directory of program library UEB.BIBL.
- (08) Directory entry of program library UEB.BIBL.
- (09) LMS is terminated.

### LMS in interactive/batch mode

LMS runs in both interactive and batch mode. In interactive mode, LMS normally reads the statements from the terminal by means of the WRTRD macro. In the course of the LMS run, statement input can be switched to the system file SYSDTA or to a library member of type J (S in the case of a source program library) by means of CTL.

Prior to the LMS run, switch 1 (/MODIFY-JOB-SWITCHES ON=1) must be set if LMS is also to read the LMS statements from the procedure when called in a procedure. Otherwise LMS will be invoked but will wait for statements from the terminal.

In batch mode, LMS reads the statements from the system file SYSDTA. CTL can be used to switch statement input to a library member of the type J (S in the case of a source program library).

If a library is still closed, LMS issues the following message up to 100 times in batch mode:

```
FILE (ELEMENT or TYP) IS LOCKED.NEXT ATTEMPT AFTER 6 SECONDS!
```

After 100 attempts control automatically passes to the next program step.

The LMS log is output to system file SYSOUT (i.e. the terminal in interactive mode) or to the medium defined by means of PRT (system file SYSLST or library member). If LMS is to output positive acknowledgments as well as error messages, processing operand PAR LOG=MED must be set.





## Definitions and conventions

### What is a library?

A library is a file with a substructure. It contains members and a directory (table of contents, TOC). Each new member added is automatically entered in the directory.

A member (also referred to as "element" in examples and messages) is a logically related set of data, e.g. a file, a procedure, an object module or a source program. Each member of a library can be referenced individually.

Storing a number of files as members in a library decreases the burden on the system catalog since each library has only one catalog entry. Storage space is saved because the members are always stored in compressed form in the library. Furthermore the members may also be stored as delta members (see section on "Filing members using the delta method", page 58).

If object modules from the EAM area are stored, the source programs need not be recompiled.

Each library has a single entry in the system catalog. The user can define the name and other file attributes such as the retention period or shareability. Catalog entries and changes to them are made by the user with the aid of system commands.

### Structure of a library

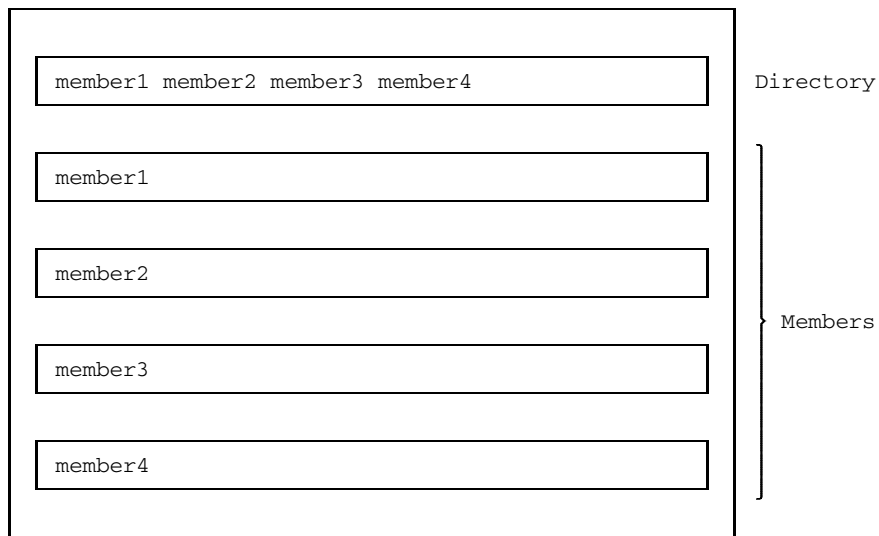


Fig. 2 Structure of a library

### Input and output libraries

LMS processes a library in the form of an input and/or output library. LMS uses the **input library** as a medium for entries, while the **output library** serves as a medium for outputting members. An input or output library is assigned by means of LIB.

LMS opens an output library for reading and writing. An input library is open for writing when DEL or NAM is used; otherwise it can only be read.

## What libraries are there?

LMS processes five library formats:

- program libraries
- source libraries
- macro libraries
- object module libraries
- sequential libraries

### Program libraries (PL)

Program libraries are PAM files that are processed with the library access method PLAM. Accordingly, they are also referred to as PLAM libraries.

The basic advantages over other library formats are that

- all member types can be stored in a single library,
- members with identical names may exist which are distinguished by type or version designation,
- the library can be accessed simultaneously by several users, even in write mode.

**Several member types in a single library**

Program libraries may contain any LMS-supported member types.

The member type indicates how the contents of members are to be interpreted by LMS and what storage unit the member belongs to:

Type	Contents of member
S	Source programs
M	Macros
R	Object modules
J	Procedures
P	Edited data
C	Load modules
D	Text data
X	Data of any format
H	Compiler result information
L	Link and load modules (LLMs)
F	IFG format masks
U	IFG user profiles

The program library features permit all data associated with a project, from source program, through object and load modules, compilation procedures and test data to documentation, to be stored in the appropriate storage units of a library.

Members of type S, M, J, P or D can also be stored with the aid of the delta method, whereby only the differences (deltas) to the previous version are stored whenever several versions of a member are present. This helps save even more storage space. When such member versions are read, LMS merges these deltas at the appropriate locations. The user is thus always offered the complete member. In addition, hierarchical relationships can be established between members (delta sequence, delta tree).

**Several versions per member type and member name**

In program libraries, a member is uniquely defined by its type, name and version designation. Furthermore it is possible to store several versions under one member type and member name.

If the user does not specify the version to be processed, LMS takes the following actions as a standard procedure:

- In read mode  
that member is sought whose specified name is accompanied by the highest version designation. The date is ignored.
- In write mode  
the actions depend on the statement:  
  
ADD, PRT  
The member is generated or overwritten with the highest version number X'FF'. LMS identifies this version by @.  
  
COR, DUP, EDT, EDR, NAM, NUM, UPD  
The output member is given the version designation of the input member.  
  
If an identically named member is overwritten, the internal variant number is incremented by 1. This serves as a write counter.

The introduction of the delta method supplies the user with the additional option of selecting between two types of storage methods. The storage method is controlled by the BASEVERSION operand in ADD and DUP. If it is specified, the member is stored as a delta member. If it is not specified, the member is added to the library as a non-delta member.

**Multiple access to program libraries**

A library can be opened by one or more users in write mode as well as in read mode.

A non-delta member can be read simultaneously by several users; it can however be written to by one user only. When a non-delta member has been opened for writing, no other access - including read access - to this member can be performed, but access to other non-delta members of the library is possible.

With the correction functions (COR, EDR, EDT, UPD) the library is simultaneously assigned as both input and output library. In this case, a user may access the non-delta member both in read and in write mode.

Delta members can only be used by one user at a time. The container (a unit of storage in a library) is locked to other users both in read and in write mode.

Fig. 3 Multiple access to members

As a result of the multiple access options to a library a member may still exist while the directory is being listed, but be no longer in existence when it is subsequently accessed: another user has deleted it in the meantime. A listing of the library's directory (see TOC) will therefore only show the current state of the input library. The user is responsible for the logical coordination of accesses to the program library members.

**Restricting multiple access**

LMS always opens a program library with SHARED-UPDATE=YES. A /SET-FILE-LINK command with SHARED-UPDATE=NO which refers to this library will have no effect. However, the user may restrict the use of multiple access by means of the following commands:

**/SECURE-RESOURCE-ALLOCATION** command:

This command prevents any other user from accessing the library while the task within which the /SECURE-RESOURCE-ALLOCATION command is issued is executing.

**/MODIFY-FILE-ATTRIBUTES** command:

This command restricts the multiple access options as desired by assigning read and write passwords, or by using the operand USER-ACCESS=OWNER-ONLY or ACCESS=READ.

For a description of these commands, see the manual "User Commands (SDF Format)" [7].

## Type-related libraries

The following libraries accept only one member type:

- source libraries
- macro libraries
- object module libraries.

Unlike program libraries, these libraries cannot accept several members that bear the same name.

LMS does not enter any file protection attributes for libraries or members. If a retention period (RETENTION-PERIOD) is specified within the /SET-FILE-LINK command, LMS transfers it to the catalog entry when a library is created.

Parallel access to source program, macro and object module libraries is possible. They may be read simultaneously by different tasks.

A library is opened by LMS for reading and writing (OPEN=INOUT) if it has been allocated as the default output library (LIB ...,USAGE=OUT) for the LMS run or has been specified in DEL, NAM or PRT.

### Source libraries (OSM)

Source libraries are ISAM files (KEY-POSITION=5,KEY-LENGTH=8); they can only recognize member type S. In source libraries, however, procedures, listings and text data can also be stored as S-type members.

For language processor runs, source programs in libraries can be assigned as input for language processors. This is effected with the command /ASSIGN-SYSDTA TO-FILE=\*LIBR-ELEM (LIBRARY=library, ELEM=member) or, in a similar fashion, in the \*COMOPT statement.

Procedure calls can be used to start the execution of command sequences in source libraries.

### Macro libraries (OSM)

Macro libraries are ISAM files (KEY-POSITION=5,KEY-LENGTH=8); they can only recognize member type M.

The assembler takes the macros referenced in the program from the macro library. The macro library must have been assigned with the /SET-FILE-LINK command.



**Object module libraries (OMLs)**

Object module libraries are files in PAM format. They take the object modules generated by the language processors as R-type members from the EAM area.

The linkage editor and the dynamic binder loader are capable of reading members from object module libraries. Up to 3380 modules may be included. The number of CSECTs/ENTRYs/COMMONs is restricted (from 380 to 800, depending on the space occupied in the library).

The object module library cannot contain more than 32500 PAM pages.



**Sequential libraries (archive libraries)**

Sequential libraries reside on magnetic tapes. These libraries are tape files with standard labels and a block size of 2048 bytes.

LMS processes sequential libraries using the BTAM access method.

A sequential library may contain BS1000 phases, object modules, macros and source programs (or procedures and other texts). Members of the same type are grouped together in a separate library section. The sequence of the various library sections, which is mandatory, is as follows: BS1000 phases, object modules, macros, source programs.

The maximum record length of object modules, macros and source programs is 80 bytes.

The directory of sequential libraries differs from that of other libraries: each member is preceded by a block which contains the member designation.

Several members of the same type may be written to a sequential library under the same name. If the members differ with respect to their version number and/or date, they can be read individually by LMS.

Sequential libraries are subject to the following restrictions:

- They cannot be processed by DEL and NAM.
- List members cannot be added to tape libraries since they have a record length > 80 characters.  
Longer records are truncated.
- The creation of continuation tapes is not possible in BS2000.  
Continuation tapes created under BS1000 can be processed, however (see page 326).

LMS does not enter any file protection attributes for libraries or members. A retention period (RETENTION-PERIOD) specified in the /SET-FILE-LINK command is transferred by LMS into the file header label (HDR1) when LMS creates a library.

**Rules for member designations in sequential libraries**

membername	Consisting of up to 8 characters
	Character set:
	Letters : A-Z
	Special characters : \$ # @ & % - (hyphen) _ (underscore)
	Digits : 0-9
	The first character must be a letter, \$, # or @.
	Exception: BS1000 job macros.
version	Version designation, precisely three characters in length
	Character set:
	Letters : A-Z
	Special characters : none
	Digits : 0-9
	Letters can only be used as the first character.

**Using the date in member designations**

The user date, which is kept for each member, can also be used to select the member to be processed. The date can be allocated by the user when creating or renaming a member.

LMS enters the current date by default. The DATE specification causes the current date to be entered for "date".

Rules for allocation of the date for sequential libraries:

date	6 characters:	cccccc
	Meaning:	YYMMDD
		YY Year
		MM Month
		DD Day

**Processing the version number**

All correction functions (COR, EDT, EDR, UPD) increment the variant number of the corrected member by 1. This also applies to the numbering function (NUM).

## What does a program library contain?

LMS processes the following member types in program libraries:

Type	Contents of library member
S	Source programs
M	Macros
R	Object modules
J	Procedures
P	Edited data
C	Load modules
D	Text data
X	Data of any format
H	Compiler result information
L	Link and load modules (LLMs)
F	IFG format masks
U	IFG user profiles

By specifying an asterisk (\*) as a member type in DEL, DUP, LST, NAM and TOC, all member types stored in the library can be referenced. If you want to have all members of a program library listed, either TOC or \*TOC\* \*/\* must be specified.

An asterisk (\*) as a member type can only be specified when program, source, macro and object module libraries are processed. Type "\*" is not valid for tape libraries.

### Member type definition

The record length of members in program libraries may be up to 32 Kbytes (including record header).

EDT processes text members with a maximum record length of 256 bytes; EDOR processes text members with a maximum record length of 244 bytes.

#### Member type S - source programs

Source programs in libraries can be used as input to compilers and assembler for language processor runs.

#### Member type M - macros

The assembler takes the macro members referenced in the program from the assigned library.

#### Member type R - object modules

Object modules generated by the compilers or the assembler are normally stored in the temporary EAM area. LMS can be directed to write such object modules as R-type members to a program library. Alternatively, the object modules generated by the compilers or the assembler can be stored directly in a program library.

These members serve as input to the linkage editor TSOSLNK and the dynamic binder loader DBL.

### **Member type J - procedures**

In this member type BS2000 procedures and LMS statements can be stored.

CTL (see page 117) can be used to read LMS statements directly from procedure members. These members may have records of any length. However, when BS2000 procedures are used, it should be borne in mind that records exceeding 80 bytes are not supported.

The invocation of BS2000 procedures from a member of type J, directly from the library, depends on the system environment.

#### *Note*

Job switch 1 must be set if LMS is called in a DO or CALL procedure and is also to read the statements from the procedure.

### **Member type P - list members**

Edited data is referred to as a list member. The first character of the record must be a valid feed control character; this is checked on output to SYSLST.

Members of this type can be generated using ADDP (see page 91), DUPP (see page 120), EDTP/EDRP (see page 126) and PRT (see page 148).

List members are printed using LST, taking the feed control character into account, provided that PRT (LST) has been specified.

**Member type C - load modules**

A load module generated by the linkage editor TSOSLNK is normally stored in a file. LMS can be directed to write such a file as a C-type member to a program library. Alternatively, the load modules generated by the linkage editor can be stored directly in a program library.

A load module stored in the program library can be used as input to the static loader ELDE and is invoked as follows:

```
/START-PROGRAM FROM-FILE(LIB=library,ELEM=member[, (VER[SION]=version)]
```

**Member type D - text data**

Any text may be written to D-type members. The same functions are possible as with S-type members.

**Member type X - data of any format**

The X-type member can accept ISAM, SAM and PAM files.

**Member type H - compiler result information**

Members of this type will be generated by the compilers and the assembler and stored in program libraries. Further details will then be given in the respective user guides.

**Member type L - LLMs**

The linkage editor BINDER [2] stores the generated link and load modules (LLMs) in members of this type. This member type is supported as of BS2000 Version 10.0A.

**Member type F - IFG format masks**

Members of this type will in future be generated by IFG and stored in program libraries. However this member type cannot be generated by LMS.

**Member type U - IFG user profiles**

Members of this type will in future be generated by IFG and stored in program libraries. However this member type cannot be generated by LMS.

**Member designations**

Members can be addressed individually in program libraries via their member type and member designation.

The member designation consists of name, version and date, and is specified in the following form:

```
membername[/version[/date]]
```

or

```
membername[/date]
```

The specification of version and date is optional. If no value is specified for version in a statement, the member having the highest value is selected by default. If no value is entered for date in a statement, the current date is entered by default.

The member designation is specified as an operand:

```
operationx member
```

operation	Name of the statement
x	Member type
member	Member designation consisting of member name and, optionally, version and date

**Rules for member designations in program libraries**

membername	Consisting of up to 64 characters
	Character set:
	Letters : A-Z
	Special characters : \$ # @ . (period) - (hyphen) _ (underscore)
	Digits : 0-9

The characters hyphen, underscore and period must not be the first or the last character, and two identical special characters must be separated by at least one other character.

The hyphen must not be placed immediately after one of the characters \$, @, #, underscore or period. The member name must contain at least one letter or one of the special characters @, #, \$.

version           Version designation, up to 24 characters in length

Character set:  
 Letters           : A-Z  
 Special  
 characters       : . (period) - (hyphen) @ (commercial at)  
 Digits           : 0-9

The special characters period and hyphen must not be the first or last character. Identical special characters must be separated by at least one other character. The hyphen must not be placed immediately after a period.

If @ (commercial at) is explicitly specified, no other character may be specified in the version designation.

If a member designation does not contain a version specification when accessed in write mode, @ is entered by default. This "commercial at" stands for the highest possible version. If the user explicitly enters a version beginning with Vd. (d=digit), 0 is prefixed to the digit (i.e. V0d). This is to ensure that, for instance, V9.x=V09.x < V10.x.

If a member designation does not contain a version specification when accessed in read mode, the highest available version will be read.

### Using the date in member designations

The user date, which is kept for each member, can also be used to select the member to be processed. When copying or renaming a member, the date of the original member is assumed unless a new date is specified explicitly. When creating a new member, LMS uses the current date by default, provided no other date is specified explicitly.

### Rules for allocation of the date for program libraries

date               10 characters:   cccc-cc-cc  
 Meaning:           YYYY-MM-DD  
                     YYYY   Year  
                     MM     Month  
                     DD     Day

When members are transferred from other libraries, the date is converted to this format as a standard procedure.

Default value: current date.

The DATE specification causes the current date to be entered for "date".



**Processing the version and variant numbers**

Automatic version updating by means of correction and numbering functions for source libraries, macro libraries and object module libraries is no longer used for program libraries.

Instead they keep a variant number (not exceeding 4 positions, numeric), which performs the function of a write access counter. There is only one variant per member. Once variant number 9999 is reached, the member must be copied for further processing.

No version updating is performed for the program libraries; instead the variant is incremented by 1 on each write access to the member. When ADD, COR, EDT, EDR, UPD or NUM is used, the variant will therefore be incremented if this statement causes a member of the same type, name and version to be overwritten.

**Logging the member designations**

Member designations are logged as follows:

```
(type)membername/version[(variantnumber)]
```

The variant number is only kept for members in program libraries. It is set to (0001) as a standard procedure and is incremented by 1 each time ADD, NUM, UPD, COR, EDT or EDR is executed.

When TOC is executed, the date is output as well:

```
(type)membername/version[(variantnumber)]/date
```

## What do type-related libraries contain?

Type-related libraries contain only one type of member.

Type	Contents of the library member	Library
S	Source programs, procedures, logs and text data	Source library
M	Macros	Macro library
R	Object modules	Object module library

### Member type definition

Member types S and M have a record length of up to 251 bytes, member type R up to 80 bytes.

#### Member type S

This member type includes source programs, procedures, logs and text data.

Members can, however, also be referenced using member type P, D or J. Thus it is possible to distribute the members of an existing source library to the corresponding member types of a program library.

Source programs in source libraries serve as input to compilers.

If a log taking the feed control characters into account is to be printed using LST, specification of PRT (LST) and the processing operand PAR FORMAT=P is mandatory.

#### Member type M

The assembler takes the macro members specified in the program from the assigned macro library.

#### Member type R

The object modules generated by the compilers can be stored in object module libraries and serve as input to the linkage editor TSOSLNK and the dynamic binder loader DBL.

**Member designation**

The member designation permits each member of a library to be addressed individually. The member designation consists of a name, version and date, and is specified in the following form:

```
membername[/version[/date]]
```

or

```
membername[/date]
```

The specification of version and date is optional. If, in a statement, no value is specified for "version", the member having the highest version is selected by default. If, in a statement, no value is specified for "date", the current date is entered by default.

The member designation is specified as an operand:

```
operationx member
```

operation	Name of the statement
x	Member type
member	Member designation consisting of member name and, optionally, version and date

**Rules for member designations in source libraries, macro libraries and object module libraries**

membername	<p>Consisting of up to 8 characters</p> <p>Character set:</p> <p>Letters : A-Z</p> <p>Special characters : \$ # @ - (hyphen) _ (underscore)</p> <p>Digits : 0-9</p> <p>The first character must be a letter, \$, # or @.</p>
version	<p>Version designation, exactly three digits</p> <p>Character set:</p> <p>Letters : A-Z</p> <p>Special characters : none</p> <p>Digits : 0-9</p> <p>Letters can only be specified as the first character.</p>

**Using the date in member designations**

The user date, which is kept for each member, can also be used to select the member to be processed. The date can be allocated by the user when creating or renaming a member.

As the default value, LMS inserts the current date. The DATE specification causes the current date to be entered for "date".

Rules for allocation of the date for source, macro and object module libraries:

```
date           6 characters: cccccc
                Meaning:      YYMMDD
                        YY     Year
                        MM     Month
                        DD     Day
```

**Processing the version number**

Source, macro and object module libraries:

All correction functions (COR, EDT, EDR, UPD) increment the version number in the corrected member by 1. This also applies to the numbering function (NUM).

## Multiple selection of member designations

The member designation references exactly one member; it may however be defined as variable in order to select several members for processing. This member designation is called multiple selection.

To permit member designations to be defined as variable, LMS provides symbols that have the following meanings for multiple selection:

Symbol	Meaning for multiple selection
' (apostrophe)	<p>Symbol permitted anywhere in member name, version and date. Selected are all members having any character in the corresponding position in the member designation.</p> <p>The apostrophe stands for <b>one</b> single character.</p> <p>For closing apostrophes blanks may used, i.e. the selected names may be shorter than the multiple selection.</p>
* (asterisk)	<p>The asterisk as a member type:</p> <p>The specification refers to all member types of the assigned library.</p> <p>The asterisk in the member designation:</p> <ul style="list-style-type: none"> <li>- in the member name           <p>The asterisk may be combined with other characters or be used as a single character symbol. If it is combined with other characters, it must be the last character of the name. From this position of the character string onwards, the name may have any length or contents. If * is the only character, the member name may have any length or contents.</p> </li> <li>- as a version           <p>The specification refers to all versions of the member name concerned.</p> </li> </ul> <p>If * is specified as the only character for the entire member designation, the specification refers to all members that designate the highest version.</p>

Symbol	Meaning for multiple selection
< (less than) > (greater than) = (equal to) # (means not equal to)	These symbols permit limiting values for version and date to be specified when selecting the members to be processed. The appropriate symbol is placed between the slash and the version, or between the slash and the date. Together with the specified values for version and date, they restrict the selection of members to be processed.
-member	This specification is to be understood as a "minus member". It excludes from processing a member that would have been selected by means of the preceding multiple selection. "member" may also be used to specify multiple selection in this case; it must however have an identifier referencing a group of members or a member within the preceding multiple selection. There must be a comma between multiple selection and "-member".

#### *Examples of multiple selection*

- Multiple selection

- AB'C\*      All members whose names begin with AB, have any character in the 3rd, and a C in the 4th position are selected. The contents from the 5th position are freely selectable.
- "/B\*      All members having a name length of up to 3 characters and with a B in the first position of the version number, are selected.
- \*/>1983\*      All members entered since 1.1.84 are selected.
- AB/\*      All AB members of the highest version, regardless of the date, are selected.

- Multiple selection with limiting values
  - \* />402 All members having a version number greater than 402 are selected.
  - A\* //<1982\* All members of the highest version whose names begin with A and which have a date earlier than Jan. 1, 1982 are selected.
  - A\* /#B\* All members whose names begin with A and which have a version number that does not begin with B are selected.
  - AB' / =107 All members whose names begin with AB, are up to 3 characters long and have the version number 107 (equivalent to specifying AB' /107), are selected.
- Multiple selection with members for exclusion
  - AB\* , -ABC, C\* All members whose names begin with AB or with C, except member ABC, are selected.
  - L' ' , -L' ' /001 All members whose names begin with L and are up to 4 characters long, except those having version number 1, are selected.

## Construction specification for member designations

In addition to the designation of the member to be processed ("member"), some LMS statements also use an additional member designation ("memberu"), which is specified after a delimiter (,>=) in the statement. "memberu" has a different meaning for the following statements:

COM                    Member designation of the member to be used in the comparison  
                          (=memberu)

ADD, DUP, NAM    Member designation for the new member (>memberu)

These member designations can be defined as variable by means of construction specifications.

The variable positions in the construction specification are taken from the member ("member") to be processed. LMS uses the formalism of multiple selection for forming construction member designations.

The symbols specified in the member designation "memberu" have the following meanings in the construction specifications:

Symbols	Meaning in the construction specification
' (apostrophe)	The position in the construction specification identified by the apostrophe is filled with the character occupying the corresponding position in the member designation "member". An apostrophe defines <i>one</i> position in the member designation.  If the apostrophe occupies the final position of the construction specification, then no more characters will be transferred from the member designation "member".
* (asterisk)	The asterisk can only be used as the final character in the construction specification. It means that from this position all the characters from the member designation "member" are to be transferred.  If * is used as the only character, the entire member designation "member" is transferred.

All other positions remain unchanged.



*Examples of construction specifications*

A library contains 3 members with the names:

1. ABC/001
2. ABCD/234
3. ABCDE/101

Statements to be executed	Names generated by LMS
NAMS ABC>'X	<ol style="list-style-type: none"> <li>1. ABX/001</li> <li>2. not renamed</li> <li>3. not renamed</li> </ol>
NAMS AB*>XY*/A02	<ol style="list-style-type: none"> <li>1. XYC/A02</li> <li>2. XYCD/A02</li> <li>3. XYCDE/A02</li> </ol>
DUPS AB''>'X'Y/A*	<ol style="list-style-type: none"> <li>1. AXC/A01</li> <li>2. AXC/A34</li> <li>3. not duplicated</li> </ol>

*Note*

When making multiple selection and construction specifications

- different input designations may possibly be mapped to the same output designation. The differing data is overwritten in accordance with the way the OVERWRITE processing operand has been set (example: NAMx A\*>B).
- if a lexicographically higher member designation is generated by means of a construction specification, and the designation in turn conforms to the multiple selection, LMS will find this member again when the directory is sequentially processed.
- If a multiple selection occurs more than once (even beyond a sublist), only the last multiple selection will be processed:

NAMS A>B, A>C

Only A>C will be processed.



## LMS functions

This chapter gives an overview of the LMS functions. The functions are initiated by statements.

Processing operands control and affect not only the statements, but also the LMS run as a whole. Job switches which are set when LMS is invoked likewise affect the LMS run. The LMS statements are described on page 81 ff, the processing operands on page 203 ff.

After LMS has been invoked, all processing operands are set to default values. If other processing operand values are required for certain statements, the processing operand must be set before entering the statement.

If the execution of each LMS statement (positive acknowledgment) is to be logged in addition to error messages, processing operand LOG=MED or LOG=MAX must be set.

Libraries cannot be processed until they have been assigned. The libraries that are assigned may already be in existence, or they may be newly created. Not until the assignment is successful can new members be entered and/or processed.

## Library assignment

All LMS functions require either an input library or an output library, or both. LMS reads members from the input library and writes members to the output library.

The LMS statement LIB is used to assign a library as an input library, an output library, or both. This assignment is valid until a further LIB is issued or until the LMS run is completed.

### Library assignment via LIB

LIB defines

- which library is assigned or closed
- whether the library is assigned as an input library, an output library, or both
- whether the library is a program, source, macro or object module library
- whether the library is an existing one, or whether it must be created

Libraries are addressed in LIB via the file name of the library, the link name, or the short designation for the library.

#### Link name

If a link name is to be used for the library, a /SET-FILE-LINK command establishing the link name to the library must be issued prior to the LMS call:

```
/SET-FILE-LINK FILE-NAME=libraryname, LINK-NAME=linkname
```

#### Example

```
/SET-FILE-LINK FILE-NAME=BIB, LINK-NAME=TEST  
/START-PROGRAM $LMS  
$LIB TEST, IN  
.  
.  
.
```

Library A.BIB is assigned the link name TEST; during the LMS run it is referenced via this link name and opened as an input library.

### Short library designation

The short library designation for a statement can be used to assign another input library temporarily. This assignment is valid for this statement only, afterwards the LIB assignment reapplies.

Libraries assigned with LIB are regarded as input or output libraries until new libraries are assigned. If no other library is assigned, the assignments apply until the end of the LMS run.

However, LMS also allows another input library to be opened for a statement. In this case the short designation of the library is specified as an operand in the statement. The validity of this library assignment expires after execution of this statement.

In the next statement which does not contain a short library designation as an operand, the input library assigned with LIB resumes its validity.

The short library designation is declared in a /SET-FILE-LINK command using the link name LIBlib:

```
/SET-FILE-LINK FILE-NAME=libraryname, LINK-NAME=LIBlib
```

- For "lib" the three digits representing the short designation in the LMS statements are used.
- Leading zeros may be omitted in statements, but not in the /SET-FILE-LINK command.
- in LMS statements, "lib" must be enclosed in parentheses.

#### Example 1

```
/SET-FILE-LINK FILE-NAME=BIB, LINK-NAME=LIB001  
/START-PROGRAM $LMS  
$LIB (1),OUT  
.  
.  
.
```

Library B.BIB is assigned a short library designation (1); during the LMS run it is referenced via this short designation and opened as an output library.

### Example 2

```
/SET-FILE-LINK FILE-NAME=EIN.BIB, LINK-NAME=LIB001 }
/SET-FILE-LINK FILE-NAME=AUS.BIB, LINK-NAME=LIB002 } _____ (01)
/SET-FILE-LINK FILE-NAME=PLA1.BIB, LINK-NAME=LIB003 }
/SET-FILE-LINK FILE-NAME=PLA2.BIB, LINK-NAME=LIB004 }
/START-PROGRAM $LMS _____ (02)
$LIB (1), IN }
$LIB (2), OUT } _____ (03)
$DUPS ELEM1>ELEM01 }
$COMS ELEM3=ELEM4 } _____ (04)
$LSTS ELEMENT1 (3) }
$TOCS (4) } _____ (05)
$LSTS ELEM1 }
$COMS ELEM1=ELEM3 } _____ (06)
$END _____ (07)
```

- (01) The I/O libraries are assigned with the /SET-FILE-LINK command since they are referenced during the LMS run by way of short designations.
- (02) LMS is invoked.
- (03) The libraries to be assigned as standard libraries for the statements are assigned with LIB.
- (04) For these statements, which contain no short designation, the libraries assigned with LIB apply.
- (05) Member ELEMENT1 from library PLA1.BIB, for which the short designation (3) has been defined in the /SET-FILE-LINK command, is listed; the directory of the library PLA2.BIB for S-type members is output.
- (06) These statements again refer to the libraries assigned with LIB.
- (07) LMS is terminated.


For a description of the statements in alphabetical order see page 91 ff.

**Sequential libraries**

Sequential libraries must be assigned with the /SET-FILE-LINK command. In the /SET-FILE-LINK command, the operand ACCESS-METHOD=BTAM is mandatory. If the tape library does not yet have a catalog entry and is to be read, an IMPORT-FILE command must be issued prior to the /SET-FILE-LINK command:

```
/IMPORT-FILE SUPPORT=TAPE(VOLUME=vsno, DEVICE-TYP=device, FILE-NAME=filename)
/SET-FILE-LINK LINK-NAME=LIBlib, FILE-NAME=filename, ACCESS-METHOD=BTAM
```

Sequential libraries cannot be created with the aid of LIB; they must be assigned with LIBOUT and the short designation (see page 330).



## Processing of members

The following sections provide an overview of the options for processing members with LMS:

LMS permits members to be

- entered in libraries as non-delta and delta members
- output to files
- output to other libraries (duplicated)
- listed
- deleted
- numbered
- compared
- renamed
- edited
- corrected

LMS also enables the library's directory to be output.

All statements mentioned in this chapter are described in alphabetical order starting at page 91 ff.



### Adding members to a library

The following statements output members to the assigned output library: ADD, COR, DUP, EDR, EDT, NUM, PRT and UPD.

The OVERWRITE processing operand determines whether or not an identically named member in the output library is overwritten.

**ADD** adds files, modules from the EAM area, members from FMS libraries, and records from the LMS statement stream to the assigned output library as members. This statement enables the user to additionally define whether the member is stored as a non-delta member or as a delta member.

Program libraries permit files with a RECORD-SIZE of up to 32 Kbytes to be stored.

If a sequential library (tape library) is assigned, the ADDC statement converts BS2000 program files to BS1000 phases and stores them as C-type members.

If an ISAM file is added, the KEY processing operand determines whether the ISAM keys and other file attributes are included. ISAM keys having a length of up to 255 bytes may then be entered.

Adding the ISAM keys to a member is especially useful for archiving.

If processing operand KEY has been set, it is also possible to include files with RECORD-FORMAT=FIXED; if not, only RECORD-FORMAT=VARIABLE is allowed.

#### *Notes*

- The ISAM keys of a source program file should not be included in the member, since the compiler cannot translate the source program from this member without errors if ISAM keys are present.
- If system file SYSDTA is assigned to a member which has stored the ISAM key, the ISAM keys are also read. The ISAM keys must then be removed from the program which carries out the processing.

Fig. 4 Adding members with ADD

**PRT** can be used to write the LMS log to a list member.

Fig. 5 Adding members with PRT

**DUP** duplicates members from the input library and outputs them to the output library, storing them there with different member designations, if desired. This statement enables the user to additionally define whether the member is stored as a non-delta member or as a delta member.

Fig. 6 Adding members with DUP

**EDT** and **EDR** branch to EDT and EDOR respectively in order to process the specified member of the input library. When EDT or EDOR is terminated, the processed member is output to the assigned output library, in some cases with a new name.

When no operands are specified for EDT or EDR, a branch is made to EDT or EDOR but no member is read into the output area.

**Outputting members**

The members of an input library can be output

- to files or FMS libraries by means of SEL
- to the output library by means of DUP.

Fig. 7      Output of members

**Listing members**

The listing of members is controlled by **LST** and **PRT**. LST defines the members and the library whose members are to be output.

PRT defines the output medium.

Processing operands can be used to control the format and size of the output.

## Deleting members

DEL deletes one member, several members or all members of an input library (program, source, macro or object module library).

In the case of program libraries, a distinction is made between logical and physical deletion:

- Logical deletion  
The entries in the directory are deleted and storage space for the member concerned is freed.
- Physical deletion  
In addition to logical deletion the storage space of the corresponding member is overwritten with binary zeros.

A member of a program library is physically deleted if the processing operand DESTROY=YES has been set or if the member of a program library contains a code indicating physical deletion. Delta members are not deleted physically until the last delta member of a delta tree, i.e. the complete delta tree, has been deleted.

## Numbering records using record numbers and check fields

Some LMS functions, e.g. correction of members, make it necessary to access specific records.

To do this, LMS offers two criteria:

- record numbers
- record IDs and check fields

### Record numbers

The record number is the position of the member record in relation to the beginning of the member. When a member is displayed on SYSLST the record number is output to the member or it is included in the comparison log (see page 51). It is specified as #number. "number" is a positive integer of up to 8 digits.

The number of the first member record is #1; the number of the second record is #2, etc.

#### *Example of a member listing*

```
#1>AAAAAAA  
#2>BBBBBBB  
#3>CCCCCC
```

**Check fields**

LMS is capable of defining and processing check fields in records to be processed. The contents of the check field are referred to as the record ID.

Record IDs define individual records, e.g. during the correction of members (see page 53), or during the evaluation of the comparison log (see page 51).

A record ID may be

- an alphabetic character string
- a numeric character string
- a combination of both
- blanks.

When records are read from an input library or file,

- a check field may be defined (processing operand CHECK), and
- its record ID checked for ascending sequence (processing operand CHECK).

When records are written to a member,

- a check field may be defined (processing operand RANGE),
- the record ID can be generated using character strings (processing operand STRING) and ascending numbering (processing operand VALUE),
- the ISAM key can be entered as the record ID (processing operand STRING) if the records belong to an ISAM file.

Fig. 8 Check field structure

If members already exist, NUM can be used

- to generate a new check field with a record ID, or
- to renumber an existing check field.

The structure of the check field is controlled by the processing operands RANGE, STRING and VALUE.

If a member is output to an ISAM file, the ISAM key can be taken from the check fields of the member records (processing operand CHECK), if no ISAM keys are stored in the member.

## Comparing members

**COM** is used to compare text members. The members are compared one record at a time. The range of the comparison operation can be defined with the aid of processing operands.

LMS compares two members, the member specified in COM before the comparison operator being defined as the primary member; the one after the comparison operator as the secondary member. The comparison operator is represented by the character "=".

The differences thus established can be logged if requested. This log is referred to as a comparison log. Following the comparison log, LMS issues the comparison statistics in the form of a table showing the results of the comparison in terms of numbers.

The comparison range must be defined with the COMPARE processing operand, before COM is issued.

In order to decide whether two records match, a distinction is made between

- formal and
- logical comparison.

In logical comparisons, blanks are ignored.

In formal comparisons, all record characters are compared.

The results of the two comparison modes (formal and logical) are logged in the same way.

To permit comparison, two algorithms are provided, namely the Heckel algorithm (standard feature as of LMS Version 1.3A) and, optionally, the cross comparison.

Experiments with the Heckel algorithm have shown that CPU time is about the same (for small-scale and medium-scale member sizes) or even lower (for large members) than when cross comparison is used. The results of the Heckel algorithm with respect to the number of SAME lines are better. When using formal comparison, the Heckel algorithm clearly outdoes the cross comparison. This advantage is important where the processing of delta members is concerned.



### Comparison log

Normally, the compared parts of the records are logged and not the complete records.

In the comparison log the results are based on the comparison of two ranges, the secondary member being taken as the point of reference, i.e. the primary member is interpreted as the new member and the secondary member as the old one.

Comparison result	Meaning
SAM (same)	The compared parts of the two records in the primary and secondary members match.
DEL (deleted)	The record with this comparison range only occurs in the secondary member.
INS (inserted)	The record with this comparison range only occurs in the primary member.

If, in the COMPARE processing operand, a number is specified for the synchronisation counter, LMS will switch to cross comparison. The **PAR COM=** statement makes it possible to return to the Heckel algorithm.

Setting the processing operand **PAR COMPARE=/COR** causes the comparison log to be converted into correction statements (see page 53, "Correction statements from the comparison log").

**Comparison statistics**

The comparison statistics supply the following information about a comparison:

- total number of records compared in primary and secondary members
- number of records inserted
- number of records deleted
- number of identical records.

In addition the result of the entire comparison is indicated:

Comparison results	Meaning
S (same)	No differences were found during the comparison.
C (changed)	Differences were found during the comparison.
I (inserted)	The secondary member was not found.
D (deleted)	The primary member was not found.
ERR (error)	An error occurred during the comparison.

Setting the processing operand PAR SUM=YES causes the comparison statistics to be stored.

Comparison statistics thus stored can be

- listed with the aid of SUMPRT,
- added to a sum field with SUMADD, and
- deleted with SUMDEL.

## Correcting members

LMS provides various correction statements for correcting members:

- COR, EDR and EDT correct text members (member types S, M, J, P, D, X)
- UPD corrects object and load modules (member types R and C) and LLMs (member type L)

### Correction of text members

The member specified via **COR** is modified with the aid of COR subfunctions, called correction statements. These correction statements, which can access one or more member records by way of record numbers or record IDs, make the following changes in the member:

- insert records
- delete records
- replace records
- change records.

### Correction statements from the comparison log

When S-type or M-type members are compared using COM, the processing operand COMPARE defines whether the comparison log is converted into correction statements.

These correction statements, which at the same time establish the differences between the members that have been compared, are written to the system file SYSOPT. If SYSOPT is assigned to a file, this file can then again be included as a procedure member.

By reassigning the statement entry by means of CTL (see page 68), the user can transfer control of the LMS run to the procedure member. The changes in the secondary member are thus performed automatically.

This LMS function enables the user to store only one version of a member. All more recent versions can then be compared with this version, and the differences can be stored in a procedure member.

**EDT and EDR statements**

**EDT** and **EDR** invoke the respective editors EDT and EDOR as a subroutine. The member specified is then processed with the aid of EDT or EDOR statements. After the editor is terminated, the corrected member is written to the output library.

EDT or EDR without a member type and operand specification results in a branch to EDT or EDOR without a member being read into the work area.

**Correction of object and load modules and LLMs**

**UPD** corrects the specified member of the assigned input library. The corrected member is then written to the assigned output library. In this case the member may be given a new member designation.

UPD has various substatements for correcting object and load modules and LLMs. These substatements are read from the statement stream directly after UPD until \*END is encountered.

The substatement functions for object modules (member type R) are as follows:

- correct text records
- cancel text corrections
- convert corrections, i.e. either REP records into text corrections or vice versa
- insert REP records
- insert INCLUDE records
- change control section attributes
- exclude record types from the input member
- rename symbols
- define control numbers
- define identifications
- define base address

The substatement functions for load modules (member type C) are as follows:

- correct text records
- cancel text corrections
- delete correction journal records
- define control numbers
- define identifications
- define segments
- define base address

The substatement functions for LLMs (member type L) are as follows:

- correct text records
- cancel text corrections
- delete correction journal records
- define identification

### Renaming members

**NAM** renames the specified members of the assigned input library. This statement also permits the renaming of members whose designations do not conform to LMS conventions.

Renaming of delta members is not permitted for audit reasons.

### Outputting library directory

**TOC** logs the directory entries of the specified members or of the entire input library.

Processing operand SORT is used to determine whether the directory remains unsorted or is to be output sorted by name, version number, date or reference name. Unless otherwise specified, member designations are output sorted by name, version number and date.

Directories of program libraries are always output sorted alphabetically by member type and name.

TOC or TOC\* \*/\* must be specified to ensure that LMS outputs the complete directory of a program library.

## Storing and calling procedures

### Storing procedures

LMS allows the user to store BS2000 procedures and ENTER procedures as members in libraries (member type J for program libraries, member type S for source libraries).

Existing procedure files can be incorporated as members into libraries by means of ADD.

Storing procedures in this way, especially where small command files are concerned, saves storage space. The number of catalog entries is decreased.

Note however that any ISAM keys that have been stored, using PAR KEY=YES, in members of program libraries, must first be removed from these members before the procedure is called.

A library member can also be assigned as the system input file (SYSDTA):

```
/ASSIGN-SYSDTA TO-FILE=(LIB=libraryname, ELEM=member [, VERSION={vers} ] [ , TYPE={type} ] [ *STD ] )
```

where \*STD means the highest available version for VERSION and member type S for TYPE. Other TYPE options are D and M.

### Calling procedures

When calling procedures that are stored as members in a library, the user must specify the library name and member name instead of the file name:

```
/DO } libname(member)
/CALL }
/ENTER }
```

*Caution*

Valid member names are not always permitted as file names.

When a library member is called, a temporary copy of the member (SAM file) is created by SYSDFILE management under the name

`S.IN.libname.membername.tsn.HHMMSS`

The library name is truncated to the first twenty characters if it is longer than twenty. HHMMSS is the time in hours, minutes and seconds. This file is erased when it is no longer used.

*Note*

Job switch 1 must be set if LMS is invoked in a DO or CALL procedure and LMS is also supposed to read the statements from the procedure.



## Filing members using the delta method

Two methods are available for storing multiple versions of one member name in program libraries:

- the non-delta storage method
- the delta storage method

The **non-delta storage method** is used to store exactly one member, i.e. all records of a member, in its own container (a unit of storage in the library). If another version is added under this name, all records of this member will also be kept in an individual container. Any relationship that may exist between these members is unknown to LMS. Such members are henceforth referred to as non-delta members.

During processing, e.g. reading a non-delta member, LMS can directly access all records of the member specified and perform the action. This method applies to all member types and, because of its fast access features, it is particularly suitable for members that are still being developed or subject to change.

For text-oriented member types S, M, J, P and D, the **delta storage method** can be used to store multiple versions of one member. With this storage-saving method only the temporarily first member is stored in its entirety in its own container. When other versions of the same member are added, then only the records that are different from those of the previous member are identified and inserted (comparison of members). In addition, the link between new version and previous version reveals the logical relationship between the individual members.

Such members are henceforth designated as delta members.

During processing, e.g. reading a delta member, the relevant records of the referenced member are filtered out. This may slow down the entire action if a great number of related delta members are present. The delta storage method is therefore specially suited for the efficient and transparent filing of member versions.



### Delta as a storage form and organizational aid

Delta members can be distinguished from non-delta members not only on account of their efficient storage form but because of the unique relationship that exists among delta members.

Storage space is saved due to the delta structure:

- Redundant records of a member with respect to the predecessor member can be identified and will not be stored again.
- For records to be identified as redundant a formal comparison is made between the records of the new version and the specified base version.

Unique relationships are established via:

- Unique member names  
Delta members that are interrelated have the same member name and thus form a range of names. For this reason, non-delta members and delta members must have different member names, which means that
  - a non-delta member is created only if no delta member exists that has the same name,
  - a delta member is created only if no non-delta member exists that has the same name.

All delta members having the same name form exactly one logically structured "delta tree" which in its simplest form is a "delta sequence".

One name is associated with exactly one delta tree.

- Unique version designations  
When a member is included as a delta member, a specification is required as to what member was the predecessor member, i.e. what member is to be used as a basis for comparison.

The conventions for member names and version designations that are established when members are added cannot be altered afterwards, for reasons of auditing and consistency. A new delta tree, if required, may be created by duplicating and simultaneously renaming the existing delta tree.

**Adding delta members**

Delta members may be added using **ADD** or **DUP**. The BASEVERSION operand can be used for

- activating delta processing, and
- defining the predecessor member to which a relationship is to be established. The comparison is also made on the basis of this predecessor member.

If the operand is omitted, a non-delta member will be added; BASEVERSION must be the last operand of the statement. Furthermore splitting of a specification between BASEVERSION= and operand value is not allowed. This means that continuation lines are not possible.

The operand values of the BASEVERSION operand permit the following distinctions to be made:

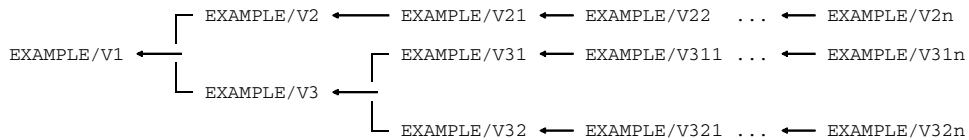
- BASEVERSION=\*NONE  
The delta member has no predecessor. This member is the first member of a delta tree. It must have a unique name. All records are stored in a new container.
- BASEVERSION=\*HIGH  
The predecessor is a delta member having the same name and the highest version designation. The new delta member is appended to the currently highest version. The delta quantity is established and stored in the container. The version designation of the new delta member should be higher than the currently highest designation.

*Example of a delta sequence (the arrows represent the relationships)*

EXAMPLE/V1 ← EXAMPLE/V2 ← EXAMPLE/V3 ... ← EXAMPLE/Vn

- BASEVERSION=version  
The predecessor member is the delta member having the same name and the specified version designation. This permits tree-like structures to be built. The new delta member is appended (quasi-laterally) to the specified version. The delta quantity is established and stored in the container. The relation to the predecessor member is to be obtained from the version designation of the new delta member.

*Example of a delta tree (the arrows represent the relationships)*



## Overview of delta members

When overviews of delta members are generated, the **TOC** processing operand has the following effect:

- If TOC=F, the previous output is extended to include the FLAG field. The D entry denotes that the member is a delta member.
- If the operand value TOC=D is specified, then a complete delta tree will be listed no matter what member has been specified via TOC.  
As well as the member designations the internal delta numbers of the members are output to field DELTA# and the internal numbers of the predecessor members are output to field BASE#. The delta number reflects the chronological order in which the members have been included. Both numbers uniquely describe the chaining of members in a tree; they cannot be controlled by the user. The user can visibly define the chaining of members by proper version specification.

## Deleting delta members

A distinction should be made between logical and physical deletion:

- Logical deletion  
This merely deletes the entry from the directory. The records and the relationship with the predecessor member remain intact.
- Physical deletion  
Controlled by the DESTROY processing operand (it must be specified when the member is included) the storage place allocated to the member is overwritten with binary zeros.  
When used for delta members this operand will only become active once the last member of the delta tree is deleted. Specification of DESTROY=YES for a member suffices to delete the entire container.

### Locking delta members

When processing

- a **non-delta member**, the specified member will be locked (member = container);
- a **delta member**, all members stored in the referenced container will be locked (delta tree = container).

In practice this means that a container can only be used for input or output.

### Restrictions when using the delta method

Here are some details that should be considered when filing members by means of the delta method:

- **Renaming** single delta members or an entire delta tree is not possible as this would disrupt the auditability of an archive or put the consistency of data at risk if a current action is somehow aborted.
- **Overwriting** delta members in the sense of 'replacing existing members' is not allowed, i.e. the OVERWRITE=YES operand is ineffective and is rejected with an error message.
- **Correcting** a delta member is not possible.

## Controlling the LMS run

The LMS run is controlled by means of the

- processing operands
- job switches.

### Effect of processing operands

The different values set in the processing operands affect the execution of LMS and its individual functions. The operands are set with the aid of **PAR**.

The following table shows which processing operand can be used to modify a particular statement or affect the entire LMS run.

Processing operands	LMS statements																	
	A D D	C O M	C O R	D E L	D E U P	D E R	E D D	E L T	N S A	N A M	N U M	P R T	R S T	S E L	S U M	T O C	U P D	L R M U S N
B[ASE]								*										
CH[ECK]	*	*	*				*	*			*			*				
COM[PARE]		*																
CS[ECT]								*										*
DES[TROY]	*		*	*	*	*	*		*	*	*							*
E[RRCONS]																		*
FC[BTYPE]														*				
FO[RMAT]								*										
I[NFO]								*										
K[EY]	*																	
LC[ASE]																		*
LIN[E]		*						*			*					*		*
LO[G]			*							*								*
LS[T]								*										
N[EWFORM]																		*

Processing operands	LMS statements																
	A D D	C O M	C O R	D E L	D E P	E D R	E D T	L S T	N A M	N U M	P R T	R S T	S E L	S U M	T O C	U P D	L R M U S N
O[VERWRITE]	*		*		*	*	*		*	*	*		*			*	
PA[TH]								*									*
PH[ASE]												*					
RA[NGE]	*		*			*	*			*							
RE[FERENCE]				*	*			*								*	
SE[GMENT]								*									
SL[ICE]								*									*
SO[RT]																*	
STRIN[G]	*		*			*	*			*			*				
STRIP					*												*
SU[M]		*															
TER[MINATE]																	*
TES[T]												*					*
TO[C]															*		
TY[PE]	*	*	*	*	*	*	*	*	*	*			*		*	*	*
V[ALUE]	*		*			*	*			*							

The processing operands define

- which member type is predefined (TYPE)
- whether members to be deleted are physically deleted (DESTROY)
- whether the ISAM key and other file attributes of ISAM files are to be included (KEY)
- with which FCB type output files of text members are created (FCBTYPE)
- the values of check fields and their interpretation (CHECK, RANGE, STRING, VALUE)
- the execution of comparisons and their interpretation (COMPARE, SUM)
- the overwriting of members (OVERWRITE)
- the scope and format of output during listing and logging (LINE, BASE, FORMAT, INFO, LOG, NEWFORM, LST, PATH, SLICE, CSECT)
- in the case of R-type or C-type members, which record types are not to be accepted as output members (STRIP)
- the logging format of the directory of program libraries (TOC)
- whether the directory of a library is to remain unsorted or to be output sorted by version number, date or reference name (SORT)
- logging and behavior in the event of an error (ERRCONS, TERMINATE)
- whether run or test mode is activated (TEST)
- whether only those members satisfying a reference condition are processed (REFERENCE)
- the load module segment which is to be processed (SEGMENT)
- the phase format to be generated (PHASE)
- whether all entries are to be converted to uppercase (LCASE)

## Controlling log output

The LMS log may include the entered statements, their execution or abnormal termination, the assigned I/O libraries as well as lists generated, for example, when members are listed or compared.

The log may be written to SYSOUT, SYSLST or to a library member. PRT is used to specify the output medium.

If the log is written to a member, LMS creates a member of type P in the case of program libraries, and type S in the case of source libraries. If the library to which the member is written is a source library, it must not be the default I/O library of the current LMS run.

The scope and format of the log is controlled by processing operands and job switches.

If job switch 4 was set when LMS was invoked, the start and end messages of LMS are suppressed. Furthermore the LMS log is limited to the minimum (corresponding to the processing operand PAR LOG=MIN).

If job switch 8 was set when LMS was invoked, the access routine messages (AMCB, DMS) are not output.

An overview of the effects of processing operands on the LMS log is given in the following table:

Processing operand	Function
LOG	Defines whether all statements, or statements only in the event of an error, or only messages are logged.
FORMAT	Defines the record format of a member being listed.
INFO	Defines the output scope of a member being listed.
SORT	Defines sorting of the directory.
ERRCONS	Defines whether and, if yes, which messages are to be logged to system file SYSOUT if nothing else is to be logged to SYSOUT.
LINE	Defines the number of lines and columns per log page.
TOC	Defines the logging format of the directory (table of contents) of program libraries.
NEWFORM	Controls line and page feed for logs.
COMPARE	Defines the range of the comparison log.



### Positive and negative acknowledgments

If the processing operand LOG=MAX or LOG=MED is specified, the execution of each LMS statement affecting a member will be logged. If the statement is executed successfully, LMS will issue a positive acknowledgment.

If the statement cannot be executed, LMS will log a negative acknowledgment and, if applicable, a corresponding error message (for error messages see page 293 ff).

All positive and negative acknowledgments have the following format:

```
[NO] statement member[word member][cause]
```

Meaning:

NO	The statement has not been executed.
statement	Statement name
member	Member designation or file name (in ADD and SEL)
word	Keyword: AS, INTO, WITH
cause	Result: EXISTING, REPLACED, etc.

#### Example

```
/SHOW-FILE-ATTRIBUTES
PROQUELL
.
.
.
ADDS PROQUEL>ELEMPRO
      NO ADD (S)PROQUEL AS ELEMPRO, OUTPUT EXISTING
ADDS PROQUELL>ELEMPRO
      ADD PROQUELL AS (S)ELEMPRO
```

### Control of statement input

**CTL** defines the medium from which the LMS statements are to be read.

The statements can be read

- from the terminal,
- from system file SYSDDTA, or
- from a library member.

If the statements are read from a member, the member must be of type J in the case of program libraries, and of type S in the case of source program libraries.

### Controlling screen overflow

**TCH** enables the user to control screen overflow and page turning.

An overflow occurs when an entry at the terminal would cause more lines to be output than specified in the /MODIFY-TERMINAL-OPTIONS command (see the manual "User Commands (SDF Format)" [7]).

TCH specifies whether in the event of an impending overflow a time of t seconds will be waited or not, or whether "PLEASE ACKNOWLEDGE" will be output before new output appears on the screen.

Furthermore it is possible to specify whether the output is to appear on a new screen or whether roll-up mode is to be activated in the event of a function change or upon acknowledgment of "PLEASE ACKNOWLEDGE".

The PLEASE ACKNOWLEDGE message is issued in the following form:

```
PLEASE ACKNOWLEDGE (NO/TIMER/<ANY INPUT>) / INTERRUPT (NE/NS/NI) :
```

The input options have the following meanings:

NO	There is no screen overflow control, i.e. if the screen is full and further data is to be output, the screen is overwritten.
TIMER	The system waits for six seconds before displaying the next screen.
<ANY INPUT>	Any specification other than NO, TIMER, NE, NS or NI causes the next screen to be displayed.

- NE Processing of a member is interrupted. Afterwards the next member satisfying the current multiple selection is processed.  
(NE : NEXT ELEMENT)
- NS The next statement is executed. The current statement and all previously gathered statements, if any, are ignored; the next statement from the statement buffer or from the procedure member assigned with the aid of CTL is executed.  
  
Once all statements have been executed, the next statement is read from the medium valid before CTL was issued.  
(NS : NEXT STATEMENT)
- NI The next input buffer is processed. All statements currently being processed are interrupted; statements that are still in the input buffer or the assigned procedure member are ignored. If a procedure member has been assigned, the statement input is reassigned to the medium which was assigned prior to the procedure member.  
(NI : NEXT INPUT)

**Execution in run or test mode**

LMS distinguishes between two types of program execution: run mode and test mode.

In run mode, all statements are executed. In test mode, only CTL, END, LIB, PAR, PRT and SYS are executed. Instead of being executed, the remaining statements are checked by LMS as to whether

- they are formally correct
- the required libraries have been assigned,
- the member names used are permissible,
- the position and length of check fields for correction functions and the numbering are correct.

In test mode, LMS calculates the control numbers for corrections made with UPD.

Thus, test mode permits errors to be detected before a function is executed.

Run mode is activated by default. Test mode is activated and deactivated by means of the TEST processing operand.

In addition, LMS switches over to test mode if a specific error occurs. The TERMINATE processing operand defines which errors cause the termination code to be set, resulting in the switch to test mode. As the default, LMS terminates run mode only in the case of fatal errors. RST is used to return LMS to run mode.

**User interfaces**

LMS enables the user to branch to a user program during the listing or comparison of members.

This subroutine can perform the following actions prior to the processing of a member record:

- manipulate the current member record
- insert own records before the current member record, or at the end of the member
- exclude the current member record from processing.

For details see page 196, USE.

## Interrupting the LMS run

### User interrupt

The user can interrupt the LMS run by pressing a program interrupt key (e.g. K2).

Continuation of the LMS run can be controlled by the INTR command, which may optionally be supplied with an input text. This input text is subsequently interpreted by LMS during interrupt handling. The current function is informed of the type of termination.

The following actions are taken when an interrupt occurs:

- LMS is interrupted by BREAK/ESCAPE (K2 or similar).  
If LMS is running in interactive mode, the appropriate STXIT routine will be activated.
- LMS executes a BKPT macro (in interactive mode only).
- Entering an INTR command causes control to be removed from the BREAK-STXIT routine and the INTR-STXIT routine to be activated.
- LMS analyzes the text supplied with the INTR command, and informs the current function about the type of termination that is desired.
- Control is removed from the INTR-STXIT routine by means of the EXIT macro.
- The interrupted function terminates in the requested form.

### INTR command

Termination of the current function is controlled by means of the INTR command, which may optionally be supplied with a text that is passed on to the interrupted function during interrupt handling. INTR commands include:

**INTR NI**            The next input buffer is processed (NEXT INPUT). All statements waiting to be executed under LMS are ignored. This means that statements currently being processed are interrupted, and that statements that are still in the input buffer, or statements that are still in the input member, are ignored. Input is reassigned to the medium that was valid before CTL was issued.

All activities that are pending are ignored.

INTR [NS]	<p>The next statement is executed (NEXT STATEMENT). The current statement and all previously gathered statements, if any, are ignored; the next statement from the statement buffer or from the CTL member is executed.</p> <p>Once all statements have been executed, the next statement is read from the medium valid before CTL was issued.</p> <p>NS is also the default value when the INTR command is entered without any operands.</p>
INTR NE	<p>Processing of a member is interrupted (NEXT ELEMENT). Processing continues with the next member that satisfies the current multiple selection.</p>

### LMS interrupt caused by errors

Error handling is also controlled by means of the STXIT routine.

Program termination occurs in the event of

- program errors ("P errors", SVC errors)
- ABEND command (abnormal end) owing to EXECUTE, LOAD, CANCEL, LOGOFF or line loss
- TIMEOUT (program/task runtime elapsed)

In all these cases a check is performed to ensure that the LMS libraries remain consistent. Basically, LMS makes sure that libraries are closed in an orderly manner.

The following applies to all program termination conditions:

- All STXIT routines in LMS are deactivated in order to prevent any incorrect continuation of processing by INTR.
- LMS simulates an END. This causes all open libraries to be closed.

If any libraries are still open at program termination time, these will be closed.

**Program errors**

Before END is simulated, the following message is issued on SYSOUT:

```
PROGRAM ERROR AT loc (IW=iw)
```

where "loc" is the interrupt address and "iw" the interrupt weight.

LMS terminates with a dump.

Program error handling has the following effects:

- Diagnostic documents will be generated at all times.
- Any continuation by LMS following program errors (this also applies to other interrupts) with RESUME, which would be pointless in any case, is prevented.

**Diagnostic aids**

Setting job switch 31 causes a test condition to be set in LMS. In the event of a program error, in batch mode, LMS terminates with a dump. In this case the registers are set in the same way as they were at the time the interrupt occurred. When job switch 31 is set in interactive mode, the following question is asked:

```
DO YOU WISH A BKPT (Y/N)?
```

If the answer is Y, the registers are loaded in the way they were defined at the time of interrupt. Then a BKPT macro is issued. The INTR interrupt routine is deactivated in order to prevent any continuation of LMS by INTR.

A dump is taken in any case before a simulated END is executed and LMS is terminated.

Entering a BKPT makes sense in interactive mode only. This is done by setting job switch 31.

An important application of this function is the testing of procedures with user exits, which permit the user to enter his own routines in the LMS run.

## Using job switches

The user can influence the LMS run by means of BS2000 job switches. They must be set by the system command `/MODIFY-JOB-SWITCHES ON=(no, . . .)` before LMS is loaded.

The following job switches affect the LMS run:

### Job switch 1:

In interactive mode, standard practice is for the LMS statements to be read from the terminal using the WRTRD macro. When job switch 1 is set, the statements are read by the RDATA macro from the file that is assigned to the logical system file SYSDTA.

When LMS is invoked in BS2000 procedures, job switch 1 must be set if the statements are to be read from SYSDTA.

### Job switch 4:

When job switch 4 is set, the start and end messages for LMS are suppressed.

### Job switch 8:

When job switch 8 is set, the access routine messages (AMCB, DMS) are not output (for messages see page 317 ff).

### Job switch 9:

By setting job switch 9, the user can request additional storage space. This makes it possible to process up to 12,000 non-matching records in one comparison and to sort extensive directories contiguously with the aid of the TOC function.

### Job switch 31:

Job switch 31 permits a test condition to be set that can be used for diagnostic purposes (see page 71).

The job switches are only interrogated on initialization; any subsequent setting and resetting has no effect for LMS.



## PAM key elimination

In future, BS2000 will only support disks with fixed block size (2 Kbytes, 4 Kbytes, etc.). These fixed block sizes are not readily compatible with PAM keys, for which reason the PAM keys will be dropped. This process has been termed "PAM key elimination".

As of BS2000 V10 (for ISAM as of BS2000 V9.5) there are two different file formats on disk for SAM, ISAM and UPAM files; the previous format tied to the PAM key (PK format) and the non-key format (NK format).

The file format is defined by the BLKCTRL value. BLKCTRL can assume the value PAMKEY, DATA or NO. For details on the file formats, please refer to the "DMS Introductory Guide and Command Interface" [14].

## Library files

The distinction between PK and NK format is primarily related to DMS. This distinction is reflected in the following ways in the internal file organization of the libraries:

- PLAM

The PAM key is not required for the organization of a PLAM library. With regard to files, however, there is a difference which is represented by the BLKCTRL file attribute.

PLAM libraries need not be converted with PAMCONV when migrating between the PK and the NK environments.

- OML

In OMLs the PAM key is used for organizing the library structure and information in the library; consequently this format can no longer be offered in the NK environment.

There is no migration. PLAM libraries must replace the OMLs.

- OSM

OSMs are based on the ISAM access method. The library utility MLU makes use of this access method, for which reason it is only indirectly affected by PAM key elimination.

When migrating between the PK environment and the NK environment, OSMs must be converted with the aid of the product PAMCONV.

Member processing

**Overview**

The following diagram provides an overview of the situations which may arise when transferring data between the file and library members.  
 For members, logical information units are listed; for files, the BLKCTR value is given.  
 The arrows indicate the transfer direction.

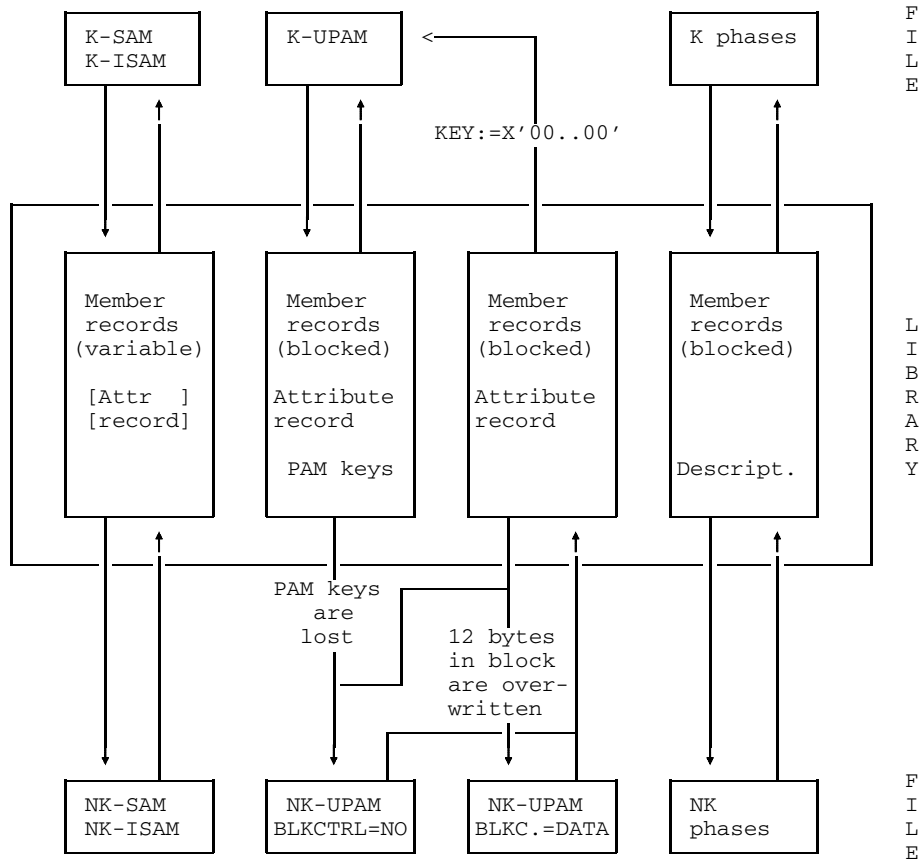


Fig. 9 Transfer of information between the file and library members

**Use of the ADD statement**

The ADD statement is used to store file contents in members as follows:

- In the case of SAM/ISAM files:

When SAM and ISAM files are added, the BLKCTRL value is also stored if PAR KEY=YES has been set, i.e. the original file block structure determined by the BLKCTRL value is documented in the attribute record.

The individual records are read using the SAM/ISAM logical access method and written unchanged to the member as variable-format records.  
The member structure generated is independent of the original BLKCTRL attribute.

- In the case of PAM files

When PAM files are added, the BLKCTRL value, too, is always stored. The blocks of the file are read using the UPAM access method and stored unchanged as blocks in the member. If PAM keys are specified, i.e. BLKCTRL=PAMKEY, these PAM keys are stored in the member.

The generated member thus retains the block structure determined by the BLKCTRL value.

- Phases

When phases are added, the BLKCTRL value is not stored. The corresponding format specification is stored on file in the phase information. In the PLAM library, PK phases and NK phases have the same format. The PAM key information is stored in descriptors.

ADD file>member	File type	BLKCTRL entry in attribute record	PAM key storage
LMS call in V10 (file on NK disk)	SAM/ISAM SAM/ISAM UPAM	— 1) from the catalog from the catalog	no no
LMS call in V10 (file on PK disk)	SAM/ISAM SAM/ISAM UPAM	— 1) from the catalog from the catalog	no for BLKCTRL=PAMKEY
LMS call V9.5	SAM/ISAM SAM/ISAM UPAM	— 1) from the catalog from the catalog (always PAMKEY)	no yes
LMS call < V9.5	SAM/ISAM SAM/ISAM UPAM	— 1) "not specified" "not specified"	no for BLKCTRL=PAMKEY

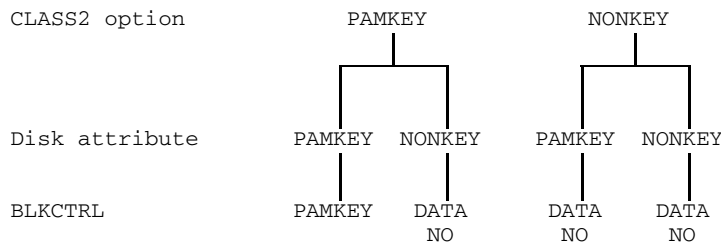
1) Storage can be controlled via PAR KEY=YES/NO

**Use of the SEL statement**

The SEL statement is used to output the contents of members to files. The BLKCTRL value is determined via the following hierarchy:

- 1) The specification in the catalog entry, FILE command or LMS parameter
- 2) BLKCTRL value stored for the member. This is relevant only for files which were originally PAM files.
- 3) Setting of the CLASS2 option to PAMKEY or NONKEY
- 4) Disk attribute PAMKEY or NONKEY

If no catalog entry exists and the BLKCTRL value has not been stored, the CLASS2 option and the disk attribute determine the BLKCTRL value:



If the CLASS2 option has been set to PAMKEY, LMS lets the system define the BLKCTRL value, i.e. BLKCTRL = <not specified>.

If the CLASS2 option has been set to NONKEY has been set, LMS sets BLKCTRL = DATA for SAM and ISAM files and BLKCTRL = NO for PAM files.

Note the following details:

– ISAM files

Variable-length member records are written using the ISAM logical access method. The BLKCTRL value of the file is determined according to the algorithm described above; in this case, however, point 2) above does not apply, as the BLKCTRL value stored for the member is used for documentation purposes only and is ignored.

– SAM files

If BLKCTRL=DATA is specified, a DMS error occurs if records in the member are longer than 32 Kbytes - 16 bytes. In the PK environment these records may have a length of up to 32 Kbytes - 4 bytes. When selecting records, LMS passes those which are too long to DMS without checking them.

The BLKCTRL value is determined in the same way as for ISAM files.

– PAM files

In the NK environment, the PAM keys are lost. In addition, when BLKCTRL=DATA is specified, the first 12 bytes of each logical block are overwritten by the system. In both cases LMS issues no error message.

– Phases (C-type members)

Phases (C-type members) are handled in a special way.

In addition to the old phase format (PK phase) there is a new PAM key free phase format (NK phase) for files. During selection, this format can be controlled with the parameter PHASE=PK/NK (default PHASE=CLASS2 option).

This parameter alone determines the BLKCTRL value and the phase format.

PHASE=PK —> Format=PK and BLKCTRL=PAMKEY

PHASE=NK —> Format=NK and BLKCTRL=PAMKEY for BS2000 <V10  
BLKCTRL=NO as of BS2000 V10

Thus NK phases can also be generated in BS2000 versions prior to V10.0 (migration aid); NK phases are only executable, however, as of BS2000 V10.

Summary

– SAM/ISAM files

It is always possible to add and select files. Any BLKCTRL values stored are used for documentation purposes only.

The internal file format is always determined by the SAM/ISAM access method. This method also converts records to the internal block format of the file.

– UPAM files

Neither the UPAM access method nor LMS can be used for the automatic conversion of data, since this would result in a loss of data.

The user has ultimate control.

File type U P A M	BLKCTRL entry generated/stored in the attribute record			
	PAMKEY	DATA	NO	—
LMS call in V10 (file on NK disk)	1)	ADD SEL	ADD SEL	— SEL
LMS call in V10 (file on PK disk)	ADD SEL	ADD SEL	ADD SEL	— SEL
LMS call < V10 (file on NK disk)	ADD SEL	— SEL	— SEL	— SEL

- 1) The value BLKCTRL=PAMKEY is not possible.
- 2) The selection process must be closed by the user, e.g. by specifying a link name in the statement.

## Statements

### Statement syntax

The following metacharacters are used for the formal representation of statements and processing operands:

Formal representation	Meaning	Example
UPPERCASE and special characters	Uppercase letters and special characters indicate constants which must be entered by the user in the specified form.	NAMx member(lib)>memberu Enter: NAMR MODLA(1)>AMOD
lowercase	Lowercase letters indicate variables for which the user must enter appropriate values.	
{ }	Braces are used to indicate alternatives, i.e. one of the enclosed entries must be selected.	<pre>PRT { (LST)       (SYSOUT)       (BOTH)       member[(lib)]       ? }</pre> <p>Enter: PRT (LST) or PRT (SYSOUT) or PRT (BOTH) or PRT PROT(1) or PRT ?</p>
[ ]	Square brackets indicate that the enclosed entries are optional.	<pre>LSTx member[(lib)]</pre> <p>Enter: LSTM MACRO or LSTM MACRO(3)</p>
...	Ellipses indicate repetition; the preceding syntactical unit may be repeated several times in succession.	<pre>member(lib),...</pre> <p>Enter: A(1) or A(1),B(2) or A(1),B(2),C(3) etc.</p>

## Statements

---

Formal representation	Meaning	Example
—	Underscoring (underline) indicates the default value. This is the value assumed by LMS in the absence of a user entry. If no value is underscored, either different values are assumed as defaults in interactive and batch modes, or no default assignment exists.	<pre>PAR TEST=[ {YES }              {NO  } ] Enter: PAR TEST=YES or PAR TEST=NO or PAR TEST= (the latter being equivalent to PAR TEST=NO)</pre>

### Statement formats

The LMS statements consist of three parts:

- operation
- operands
- comments

### General format:

```
[$]operation_operands_comments
```

The first position of a statement may contain a \$ character (except for the \$ statement); however, it is not mandatory.

### Operation

The statement must start with the operation. This consists of the statement name and - if members are to be processed by the statement - the member type. The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.

#### *Example*

ADDx	Represents the statement syntax
ADD	Statement name
x	Statement type, e.g. ADDS for source programs

The member types supported by each statement are indicated in the description of the relevant statement.



## Operands

Separated by at least one space, the operation is followed by the operands. The operands themselves are separated by commas. In some statements the delimiters "=" and ">" are also used to separate operands. The character ">" represents the arrow symbol showing the processing direction.

No delimiter may be specified before the first or after the last operand of the complete statement. No space is allowed in operands or between operands and delimiters. Statements may have a maximum length of 2028 bytes.

### *Example*

```
ADDS QUELL.DAT>QUELL.ELEM
```

The file QUELL.DAT is added to the library as member QUELL.ELEM.

## Comments

Following the operands and separated by at least one blank, comments may be added.

No comments may be entered in statements in which the operand specification is omitted.

Should comments extend over a complete line (comment lines), these lines must be identified by an asterisk in column 1 and a blank in column 2.

In the comment text the characters ! and X'15' (NEW-LINE) must not occur, as they would be interpreted as statement delimiters.

## Continuation lines

A statement may consist of one or more lines. The operation portion must appear at the beginning of the first line; the operand portion may extend over several lines.

In order to indicate a continuation, a continuation character or a space must be specified immediately after one of the delimiters. The continuation character must be between columns 1 and 72.

The statement may be continued at any position of the continuation line.

*No* continuation lines are possible in conjunction with substatements (e.g. \*COR following the UPD statement).

### **Continuation characters**

The continuation character may be represented by a hyphen "-" or by the plus sign "+". The BASEVERSION operand must not be separated from its operand value by means of a continuation character.

Within a statement, separation is possible after ",", ">" or "=".

### **Entry of blocked statements**

Statements may also be entered in blocked format.

This means that, in interactive mode, each statement need not be entered separately; instead data transfer can be started for several statements concurrently.

Thus, several statements - separated by the exclamation mark (!) - and statements extending over several lines - separated by the logical end-of-line character - can be sent off in one line.

#### *Exception:*

Statements entered in block format after a CTL statement are not executed.

The logical end-of-line character is the NEW-LINE character (NL) valid for the particular type of terminal. Normally it is represented by the character \ or <, depending on the type of terminal.

## Overview of statements

Statement	Application
<p>ADD[x] { filename, ... LINK=linkname [prefix.] (name, ...) [.suffix] } [&gt;memberu]</p> <p style="padding-left: 40px;">[, BASEVERSION={ *NONE version *HIGH }]</p> <p>ADD[R] *OMF [(module, ...)] [&gt;memberu]</p> <p>ADD[x] FMS=fmslib (fmsmember) [&gt;memberu]</p> <p style="padding-left: 40px;">[, BASEVERSION={ *NONE version *HIGH }]</p> <p>ADD[x] { (CTL) (CMD) ([SYS]DTA) } &gt;memberu [, BASEVERSION={ *NONE version *HIGH }]</p> <p>ADD[C] { filename LINK=linkname }</p>	<p>Add data to a library</p>
<p>COM[x] { member, ... [(lib)] (lib) } =memberu [(libu)] [, {...}=...]</p>	<p>Compare members</p>
<p>COR[x] member [(lib)] [&gt;memberu]</p>	<p>Correct text members</p>
<p>CTL { (CMD) ([SYS]DTA) (RDR) member [(lib)] ? }</p>	<p>Control statement input</p>
<p>DEL[x] { member [(lib)] (lib) } [, ...]</p>	<p>Delete members</p>
<p>DUP[x] { member, ... [(lib)] (lib) } [&gt;memberu] [, {...}] [&gt;...]</p> <p style="padding-left: 40px;">[, BASEVERSION={ *NONE version *HIGH }]</p> <p>DUP[x] name1 [(lib)] [&gt;name2] , STRUC=Y [ES]</p>	<p>Duplicate members and delta trees</p>

Statement	Application
EDT[x] member [(lib)] [>memberu] EDR[x] member [(lib)] [>memberu] EDT EDR EDT[x] memberu[(lib)] >*DUMMY EDR[x] memberu[(lib)] >*DUMMY	Branch to an editor;  create, correct and view text members or files
END	Terminate the LMS run
LIB $\left\{ \begin{array}{l} \text{FILE=libname} \\ \text{LINK=linkname} \\ \text{[LIBRARY=]name} \\ \text{[LID=] (lib)} \end{array} \right\} [, [\text{USAGE=}] \left\{ \begin{array}{l} \text{IN} \\ \text{OUT} \\ \text{BOTH} \end{array} \right\}]$  $[, [\text{FORMAT=}] \left\{ \begin{array}{l} \text{PL} \\ \text{OML} \\ \text{OSM} \end{array} \right\}] [, [\text{STATE=}] \left\{ \begin{array}{l} \text{O[LD]} \\ \text{N[EW]} \\ \text{A[NY]} \end{array} \right\}]$  LIB C[LOSE] [, $\left\{ \begin{array}{l} \text{FILE=libname} \\ \text{LINK=linkname} \\ \text{[LIBRARY=]name} \\ \text{[LID=] (lib)} \end{array} \right\}]$  LIB ?	Assign and close libraries
LST[x] $\left\{ \begin{array}{l} \text{member[(lib)]} \\ \text{(lib)} \end{array} \right\} [, \dots]$	List members
NAM[x] $\left\{ \begin{array}{l} \text{member, ... [(lib)]} \\ \text{(lib)} \end{array} \right\} >\text{memberu}[, \{ \dots \} > \dots]$	Rename members
NOP [string]	No operation
NUM[x] $\left\{ \begin{array}{l} \text{member[(lib)]} \\ \text{(lib)} \end{array} \right\} [>\text{memberu}]$	Number member records
PAR $\left[ \left\{ \begin{array}{l} \text{parname} = \left[ \begin{array}{l} \text{parvalue} \\ ? \end{array} \right] \right\} , \{ \dots \} \right]$	Set processing operands

Statement	Application
PRT $\left\{ \begin{array}{l} \text{(LST)} \\ \text{([SYS]OUT)} \\ \text{(CON)} \\ \text{(BOTH)} \\ \text{member [(lib)]} \\ \text{?} \end{array} \right\}$	Control log output
RST [STOP]	Quit test mode
SEL[x] member, ... [ $\left\{ \begin{array}{l} \text{[prefix.] (name) [.suffix]} \\ \text{filename} \\ \text{LINK=linkname} \end{array} \right\}$ ]	Output members to files and FMS libraries
SEL[x] member>FMS=fmslib(fmsmember)	
SUM ['text']	Store comparison statistics
SUMPRT $\left\{ \begin{array}{l} \text{[S1]} \\ \text{[S2]} \end{array} \right\}$ [, 'text']	Output comparison statistics
SUMADD $\left\{ \begin{array}{l} \text{[S1]} \\ \text{[S2]} \end{array} \right\}$ > $\left\{ \begin{array}{l} \text{[S1]} \\ \text{[S2]} \end{array} \right\}$	Add comparison statistics
SUMDEL $\left\{ \begin{array}{l} \text{[S1]} \\ \text{[S2]} \end{array} \right\}$	Delete comparison statistics
SYS [ $\left\{ \begin{array}{l} \text{'systemcommand'} \\ \text{systemcommand} \end{array} \right\}$ ]	Issue system commands
TCH $\left\{ \begin{array}{l} \text{[O[FLOW]=} \left\{ \begin{array}{l} \text{N[O]} \\ \text{A[CK]} \\ \text{T[IMER]} \\ \text{?} \end{array} \right\} \text{]} \text{[, N[EWScreen]=} \left\{ \begin{array}{l} \text{Y[ES]} \\ \text{N[O]} \\ \text{?} \end{array} \right\} \text{]} \\ \text{?} \end{array} \right\}$	Change terminal characteristics
TOC[x] [ $\left\{ \begin{array}{l} \text{member [(lib)]} \\ \text{[ (lib) } \end{array} \right\}$ ], ...]	List directory of a library
UPD[x] member [(lib)] [>memberu]	Update object and load modules and LLMS

Statement	Application
<pre> USE {   {     {LST}     {COMP} [= {       {         {entry}         {*}         {libraryu}       } (entry)     } ]   }   {     {EDTLIB}     {EDORLIB} =librarys     {FMSLIB}   }   ? }           </pre>	<p>Branch to user programs</p>
<p>\$</p>	<p>Output statement buffer</p>

**Member types in statements**

The following table shows which members are allowed in the individual statements.

Member type	S	M	R	J	P	C	D	X	H	L	F	U	*
Statement													
ADD	+	+	+	+	+	+	+	+					
COM	+	+		+	+		+	+					
COR	+	+		+	+		+	+					
DEL	+	+	+	+	+	+	+	+	+	+	+	+	+
DUP	+	+	+	+	+	+	+	+	+	+	+	+	+
EDT/EDR	+	+		+	+		+	+					
LST	+	+	+	+	+	+	+	+	+	+	+	+	+
NAM	+	+	+	+	+	+	+	+	+	+	+	+	+
NUM	+	+		+	+		+	+					
SEL	+	+	+	+	+	+	+	+					
TOC	+	+	+	+	+	+	+	+	+	+	+	+	+
UPD			+			+				+			

+: Member type valid in statement

empty field: Member type not valid for statement

**Table of required libraries**

The following table shows which libraries are required by the individual LMS statements:

Function	Required by LMS	
	Input library	Output library
ADD	no	yes
COM 1)	yes	no
COR 3)	yes	yes
CTL	yes	no
DEL 2)	yes	no
DUP 3)	yes	yes
EDR 3)	yes 4)	no
EDT 3)	yes 4)	no
LST	yes	no
NAM 2)	yes	no
NUM 3)	yes	yes
PRT	no	yes
TOC	yes	no
SEL	yes	no
SUM	no	no
SUMADD	no	no
SUMDEL	no	no
SUMPRT	no	no
UPD 3)	yes	yes

- 1) The comparison members may reside in different input libraries, but they must not be kept in the same sequential library.
- 2) The input library is opened for write access.
- 3) Input and output libraries may be identical, except where sequential libraries are concerned.
- 4) The input library is required only if an existing member is corrected with the aid of the editors.



## ADD Add data to a library

ADD permits

- files
- modules from the EAM area
- members of an FMS library
- member records from the LMS statement stream, and
- load modules (BS2000 phases) in the form of BS1000 phases in sequential libraries to be added as library members.

ADD has five different formats to perform the above functions.

### Format 1: Add files

Operation	Operands
ADD[x]	$\left\{ \begin{array}{l} \text{filename, ...} \\ \text{LINK=linkname} \\ \text{[prefix.] (name, ...) [.suffix]} \end{array} \right\} [>\text{memberu}]$ $[\text{, BASEVERSION} = \left\{ \begin{array}{l} *NONE \\ \text{version} \\ *HIGH \end{array} \right\}]$

### Format 2: Add modules from the EAM area

Operation	Operands
ADD[R]	*OMF [ (module, ...) ] [>memberu]

**Format 3:** Transfer a member from an FMS library to a LMS library

Operation	Operands
ADD[x]	FMS=fmslib(fmsmember) [>memberu] [, BASEVERSION={*NONE version *HIGH}]

**Format 4:**  
Generate a member by adding member records from the LMS statement stream

Operation	Operands
ADD[x]	{ { (CTL) (CMD) ( [SYS]DTA) } } >memberu [, BASEVERSION={*NONE version *HIGH}]

**Format 5:**  
Add BS2000 load modules as BS1000 phases to sequential libraries (tape libraries)

Operation	Operands
ADD[C]	{ filename LINK=linkname }

**Format 1:** Add files

This format of ADD adds files as members to the open output library. If the output file is a program library, a member can be stored either as a non-delta or as a delta member. The output library must previously have been assigned with LIB.

Files cataloged with RECORD-FORMAT=U can also be incorporated in libraries. Files having RECORD-FORMAT=FIXED can only be stored using PAR KEY=YES. The BLKSIZE and RECSIZE values may vary. However, the maximum record length of 32 Kbytes (including the record header) must not be exceeded. File generation groups can only be incorporated using LINK= and a valid LMS member designation.

Operation	Operands
ADD[x]	$\left\{ \begin{array}{l} \text{filename, ...} \\ \text{LINK=linkname} \\ \text{[prefix.] (name, ...) [.suffix]} \end{array} \right\} [ > \text{memberu}]$ $[ , \text{BASEVERSION} = \left\{ \begin{array}{l} * \text{NONE} \\ \text{version} \\ * \text{HIGH} \end{array} \right\}]$

- ADDx** Statement name and specification of member type.
- For non-delta members, the following member types are valid:  
S, M, R, J, P, C, D, X
  - For delta members, only the following text-based member types are valid:  
S, M, J, P, D
- The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
- filename** Fully qualified file name or multiple selection. "pathname" can also be specified for "filename" (see the manual "User Commands (SDF Format)" [7]).  
If temporary files are added, multiple selection is not allowed. Specification of more than one file name is only meaningful if an \* or a construction specification is made for "memberu".
- LINK=linkname** Link name referring to the file.

[prefix.](name,...)[.suffix]	<p>This specification permits the selection of several files that are to be added to the library.</p> <p><b>prefix</b> Common prefixed portion of names of the files to be selected. "prefix" must end with a period.</p> <p><b>suffix</b> Common suffixed portion of names of the files to be selected. "suffix" must begin with a period.</p> <p><b>name</b> Portion of a name by which prefix and/or suffix are to be supplemented to form one or more fully qualified file names. This subname is also used for the generated members added to the library if "memberu" is not specified. "name" also allows multiple selection.</p> <p>Partially qualified file names may be specified in the form prefix.(*).suffix.</p> <p>If the parentheses of (name) are omitted, ADD will act as though filename had been specified.</p>
memberu	<p>Designates the member to be created.</p> <p>When [prefix.](name,...)[.suffix] or multiple selection is specified for "filename", construction specifications are also permitted. "memberu" may be omitted, in which case the member created is given the name of the input file, without prefix and suffix.</p> <p>The various syntax rules for program libraries (see page 25) and other library types (see page 29) should be observed when defining member designations for the members to be created.</p>
BASEVERSION	<p>Defines the base member for delta storage. It refers to all members to be generated (memberu). It must be the last operand of the statement.</p> <p>If this operand is not specified, a non-delta member is created.</p>
=*NONE	<p>"memberu" is stored as the first member (=base) of a delta tree (first generation).</p>
=version	<p>Base member is the member having "version" as version designation; this member must be present and stored as a delta member.</p> <p>A construction specification is not allowed.</p>

=\*HIGH            Base member is the member having the highest version designation (at generation time of the delta member). The generated delta member is appended to this base member.

### Processing operands

TYPE	Defines the member type, if no type is specified in the statement itself.
KEY	Defines whether file attributes and existing ISAM keys are to be included.
DESTROY	Defines whether a code for physical deletion is entered in the output member (only possible in conjunction with program libraries).
OVERWRITE	Defines whether an identically named non-delta member in the output library is overwritten, not overwritten or expanded by means of new records. This processing operand cannot be used for delta members.
RANGE	Defines the position and length of the check field in the output records of text members.
VALUE	Specifies the initial value and increment for renumbering in the check field.
STRING	Defines whether the ISAM key is to be stored in the check field, or specifies the character string to be entered left-justified in the check field of output records.
CHECK	Defines the position and length of the check field in input records and checks for ascending numbering.

*Example 1*

The following files are available:

A.B.C.A    A.B.C.B    A.B.C.C    B.B.C.A    C.B.C.A

The following statements permit selection of the files indicated:

```
ADDx A.B.(*)>X.*      files:  A.B.C.A  → member name X.A
(only possible for    A.B.C.B  →→ member name X.B
program library)      A.B.C.C  →→ member name X.C
```

```
ADDx (A,C).B.C.A      files:  A.B.C.A  → member name A
                       C.B.C.A  → member name C
```

*Example 2*

```
/START-PROGRAM $LMS
$LIB TESTLIB,OUT
$ADDS S.PROG>SPROG
$END
```

The source program in file S.PROG is added to the program library TESTLIB as an S-type member with the member designation SPROG.

*Example 3*

```
/SET-FILE-LINK FILE-NAME=PROG.DAT, LINK-NAME=PROG
/SET-FILE-LINK FILE-NAME=TEST.COB, LINK-NAME=TEST
/START-PROGRAM $LMS
$LIB TESTLIB,OUT
$ADDD LINK=PROG>DDAT
$ADDS LINK=TEST>TCOB
$END
```

The files PROG.DAT and TEST.COB are added to the program library TESTLIB as members DDAT and TCOB. They are referenced in ADD via their link names.

**Format 2:** Add modules from the EAM area

This format of ADD is used to transfer modules from the EAM area of the current task and to add them as members of type R to the assigned output library. Output libraries are program libraries, module libraries and sequential libraries.

Operation	Operands
ADD[R]	*OMF [ (module, ...) ] [>memberu]

<b>ADDR</b>	Statement name with member type R. The member type need not be specified if member type R has been specified in the TYPE processing operand.
<b>module</b>	Name of a module in the EAM area, up to 8 characters in length. If the EAM area contains several identically named members, LMS transfers the module which was last compiled to the library. The module specification may be omitted. If so, LMS will transfer all modules contained in the EAM area. Multiple selection is permitted.
<b>memberu</b>	Member designation of the member to be created. A construction specification is allowed. "memberu" may be omitted. If so, modules will receive the names they have in the EAM area.

*Important notes*

- If a member designation is specified for "memberu", then for "module" the name of the module to be added must be specified if there are several modules in the EAM area.  
Otherwise, all modules from the EAM area are incorporated under the designation "memberu"; the one which was previously added will then be overwritten each time the process is performed.
- Specifying OVERWRITE=EXTEND leads to errors.

*Example*

```

/START-PROGRAM $LMS
$LIB TESTLIB,OUT
$ADDR *OMF (MOD1,MOD2)>ELM*
.
.
$END

```

The object modules MOD1 and MOD2 from the EAM area are incorporated in the output library TESTLIB as members with the names ELM1 and ELM2.

**Format 3:** Transfer a member from an FMS library

This format of ADD is used to transfer members from an FMS library (see the "FMS" manual [10]) to the open output library. If the output library is a program library, a member may be stored either as a non-delta member or as a delta member. The output library must already have been assigned by means of LIB.

This function causes FMS to be invoked internally.

Operation	Operands
ADD[x]	FMS=fmslib (fmsmember) [>memberu] [ , BASEVERSION={ *NONE version *HIGH }]

- ADDx** Statement name with specification of the member type.
- For non-delta members, the following member types are valid:  
S, M, R, J, P, D, X
  - For delta members, only the text-based member types are valid:  
S, M, J, P, D
- The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
- fmslib** Fully qualified file name of an FMS library.  
"pathname" can also be specified for "fmslib" (see the manual "User Commands (SDF Format)" [7]).
- fmsmember** Complete name of the member to be transferred or \*.  
\* serves to copy an entire FMS library.
- memberu** Member designation of the member to be created or \*.  
\* causes members to be transferred to the LMS library with the same names they had in the FMS library.
- BASEVERSION** Defines the base member for delta storage. It must be the last operand of the statement.  
If this operand is not specified, a non-delta member is created.
- =\*NONE** "memberu" is stored as the first member (=base) of a delta tree (first generation).
- =version** Base member is the member having "version" as the version designation. This member must be present and stored as a delta member.  
A construction specification is not allowed.



=\*HIGH            Base member is the member with the highest version designation (at the time of generating the delta member). The generated delta member is appended to this delta member.

*Notes*

- If OVERWRITE=EXTEND is specified, the function is aborted with an error.
- All records are expected to have variable record format.

**Processing operands**

RANGE	Defines the position and length of the check field in the output records of text members.
VALUE	Specifies the initial value and increment for renumbering in the check field.
STRING	Specifies whether the ISAM key is to be stored in the check field, or specifies the character string to be entered left-justified in the check field of the output records.
OVERWRITE	Defines whether an identically named non-delta member in the output library is overwritten, not overwritten or expanded by means of new records. This processing operand is not allowed for delta members.
CHECK	Defines the position and length of the check field in input records and checks for ascending numbering.

*Example*

```
/START-PROGRAM $LMS
$LIB LMSLIB,OUT,NEW
$ADDS FMS=FMSBIB2(SRC)>SRC1
$END
```

Member SRC from the FMS library FMSBIB2 is incorporated as member SRC1 in the program library LMSLIB which is to be created as a new library.

**Format 4:**

Create a member by incorporating member records from the LMS statement stream.

This format of ADD is used to transfer records from an LMS statement sequence to a member of the open output library. If the output library is a program library, a member may be stored either as a non-delta or as a delta member. The output library must already have been assigned with LIB. The records must directly follow ADD. The string of records must be concluded with **\*END**.

These records are input from

- the terminal
- system file SYSDDTA
- a library member.

Operation	Operands
ADD[x]	$\left[ \begin{array}{l} \{ \text{CTL} \} \\ \{ \text{CMD} \} \\ \{ \text{SYSDDTA} \} \end{array} \right] > \text{memberu} [ , \text{BASEVERSION} = \left\{ \begin{array}{l} * \text{NONE} \\ \text{version} \\ * \text{HIGH} \end{array} \right\} ]$

**ADDx**

Statement name and specification of member type.

- For non-delta members, the following member types are valid:  
S, M, R, J, P, D, X
- For delta members, only the text-based member types are valid:  
S, M, J, P, D

The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.

CTL

Records are added from the input medium defined using CTL, i.e. either the terminal, system file SYSDDTA or a library member.

CMD

In interactive mode, records are read from the terminal, in batch mode from the system file SYSDDTA.

SYSDDTA

Records are read from system file SYSDDTA.

*Note*

If records are read from the system file SYSDDTA, they must not begin with "/". The reason for this is that the RDATA macro interprets such records as commands and thus passes the return code for EOF. Therefore it is not possible to pass system commands as records.

memberu	Member designation of the member to be generated. A construction specification is not allowed. memberu is mandatory.
BASEVERSION	Defines the base member for delta storage. It must be the last operand of the statement. If this operand is not specified, a non-delta member is created.
=*NONE	memberu is stored as the first member (=base) of a delta tree (first generation).
=version	Base member is the member with "version" as its version designation. This member must be present and be stored as a delta member. A construction specification is not allowed.
=*HIGH	Base member is the member with the highest version designation (at the time of generating the delta member). The generated delta member is appended to this base member.

### Processing operands

TYPE	Defines the member type, if no type is specified in the statement itself.
DESTROY	Defines whether a code for physical deletion is entered in the output member (only possible in conjunction with program libraries).
OVERWRITE	Defines whether an identically named non-delta member in the output library is overwritten or not. OVERWRITE=EXTEND is not allowed. This processing operand is not allowed for delta members.
RANGE	Defines the position and length of the check field in the output records of text members.
STRING	Defines whether the ISAM key is to be stored in the check field, or specifies the character string to be entered left-justified in the check field of the output records.
VALUE	Specifies the initial value and increment for renumbering for text members in the check field.

**\*END Terminate transfer**

\*END terminates the transfer of records to a member.

Operation	Operands
*END	

\*END                    Name of the statement

*Example*

```

/START-PROGRAM $LMS
$LIB LIBRARY,BOTH
$ADDD >LETTER.A
* Dear ...
.
.
.
**END
$END

```

The text "Dear ...," is stored, in uppercase letters, in member LETTER.A. If lowercase letters are also to be stored in the member, the LCASE processing operand must be used (see page 225).

**Format 5:**

Add BS2000 load modules as BS1000 phases to sequential (tape) libraries

This format of ADD is used to generate BS1000 phases on a library tape from a BS2000 load module.

Operation	Operands
ADD[C]	<pre>{ filename } { LINK=linkname }</pre>

**ADDC** Statement name with the member type C.  
The member type need not be specified if member type C has been defined in the TYPE processing operand.

**filename** Fully qualified file name of a program file generated with the linkage editor TSOSLNK. "pathname" may also be specified for "filename" (see the manual "User Commands (SDF Format)" [7]).

**LINK=linkname** Link name referencing a BS2000 program file assigned with the /SET-FILE-LINK command. The file must have been generated with the linkage editor TSOSLNK.

In the sequential library the member receives the name given to the load module in the program file.

The library must be assigned beforehand with LIBOUT ...,NEWLIB (see page 330).

For further information, please refer to the section on BS2000-BS1000 compatibility on page 326).

**Processing operand**

**TYPE** Defines the member type if no type is specified in the statement itself.

*Example*

The BS2000 load module PROG.DAT is incorporated in the tape library DOS-LIB.

```
/CREATE-FILE FILE-NAME=DOS-LIB, -  
/          SUPPORT=TAPE(VOLUME=E1000A, DEVICE-TYP=T9P)  
/SET-FILE-LINK FILE-NAME=DOS-LIB, LINK-NAME=LIB001, ACCESS-METHOD=BTAM  
/START-PROGRAM $LMS  
$LIBOUT (1),NEWLIB  
$ADDC PROG.DAT  
.  
.  
.  
$END
```

The tape library DOS-LIB is created using a /CREATE-FILE command and assigned using the /SET-FILE-LINK command. It is defined as an output library by means of LIBOUT (1),NEWLIB. The module from the file PROG.DAT is transferred onto the tape and its name is retained.

## COM Compare members

COM permits members to be compared with one another. The differences thus established are listed in a comparison log and in the comparison statistics. The members may be located in different libraries. The members to be compared with one another must not be contained in the same sequential library. The COMPARE processing operand can be used to determine whether a formal or a logical comparison is made and which comparison algorithm is to be used. Moreover, it defines the comparison fields.

For further information on the comparison log and comparison statistics see page 50. Examples of COM are given on pages 262, 274 and 282.

USE permits the member records to be accessed via a user program, prior to the actual comparison.

Operation	Operands
COM[x]	$\left\{ \begin{array}{l} \text{member, ... [(lib)]} \\ \text{(lib)} \end{array} \right\} = \text{memberu}[(\text{libu})][, \{ \dots \} = \dots]$

COMx	Statement name and specification of the member type: S, M, J, P, D, X
	The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Member designation of the primary members that are to be compared with the member "memberu", or multiple selection.
lib	Short designation of the input library for "member".
memberu	Member designation of the secondary member, or construction specification.
libu	Short designation of the input library for "memberu".

**Processing operands**

TYPE	Defines the member type, if no type is specified in the statement itself.
LINE	Defines the number of lines and columns on a log page for output to system file SYSLST or to a library member.
CHECK	Only effective for cross comparison, i.e. a value between 1 and 9 must be specified in the COMPARE processing operand for the synchronisation counter. Defines position and length of the check field in the records to be compared, and checks for ascending numbering.
COMPARE	Controls the length and position of the comparison field, the type of comparison, the synchronization count, the logging mode and the generation of correction statements.
SUM	Controls the storage of comparison statistics.

The function is executed even if only one of the comparison members is found in the specified libraries. This permits counting of the records in members.

The SUM processing operand permits the comparison statistics to be stored. Stored comparison statistics can be further processed with the aid of SUM, SUMADD, SUMPRT and SUMDEL.

If job switch 9 is set, a maximum of 12,000 non-matching records may be synchronized (see page 74).

**Synchronization**

During the comparison operation LMS attempts, whenever possible, to resume on matching record sequences following non-matching record sequences. This is called synchronization. If at least as many matching records are found as required in the COMPARE processing operand, the synchronization attempt is considered to be successful. If successful synchronization was not possible, even matching records are logged as being non-matching.



*Example*

```
/SET-FILE-LINK FILE-NAME=BIBU, LINK-NAME=LIB005
/START-PROGRAM $LMS
$LIB FILE=PLIB
$PAR COMPARE=5/26/L/MAX
$COMS ASRC=ASRC (5)
.
.
.
$END
```

The different ASRC members of the libraries BIBU and PLIB are compared. The COMPARE processing operand (see page 212) is used to define the comparison range (5th to 30th byte of the member record), the type of comparison (logical comparison) and the scope of the comparison log (MAX).

For further examples see page 253 ff.

## COR Correct text members

COR corrects the specified member and outputs it to the assigned output library. The corrections are effected by way of correction statements that are expected immediately after COR.

COR processes member records having a length  $\leq 251$  bytes. Longer records are truncated, in which case LMS issues a warning message.

In the case of sequential libraries, the input library must not be identical with the output library. In the case of other libraries, the input library may be identical with the output library.

Operation	Operands
COR[x]	member[(lib)] [>memberu]

CORx	Statement name with specification of member type: S, M, J, P, D or X
	The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Member designation of the member to be corrected or multiple selection (no list specification).
lib	Short designation of the input library.
memberu	Member designation of the output member or construction specification.

### Processing operands

TYPE	Defines the member type, if no type is specified in the statement itself.
DESTROY	Defines whether a code for physical deletion is set in the output member (only possible for program libraries).
CHECK	Defines the position and length of the check field in input records and checks for ascending numbering.
RANGE	Defines the position and length of the check field in output records.
VALUE	Specifies the initial value and increment for renumbering in the check field.

STRING	Specifies the character string to be entered left-justified in the check field of output records.
OVERWRITE	Controls overwriting of identically named members in the output library. If, input library = output library and member = memberu, this processing operand will have no effect and the input member is overwritten.
LOG	Controls the scope of the correction log.

*Note*

PAM files stored in the library cannot be corrected.

During the correction process, a new member will only be created if LMS is not in test mode and if no errors were detected during correction.

If an error is detected in interactive mode, the correction process must be terminated with \*END and subsequently restarted.

In the case of program libraries (see page 27), the corrected member with its old name, variant number incremented by 1, and time of day is written to the output library; where other libraries are concerned, the version number is incremented by 1 (see page 30).

If a check field is defined in the input records (CHECK not equal to NO), an input member with ascending numbering is a prerequisite. Check fields cleared with spaces are permitted and do not lead to sequence errors. Short records with no check field are also permitted.

The correction statements must be presented in ascending order (sequential processing in the member).

If a member is simply to be renumbered, \*END following COR is sufficient.

The output mode for the correction journal is controlled by the LOG operand.

When	LOG=MIN	all changes are listed
	LOG=MED	the correction statements are also output
	LOG=MAX	all the records transferred unchanged from the input member are also logged.

Corrections can be made either via correction statements or by specification of the record ID of a data record to be inserted or replaced.

### Overview of correction statements

Correction statement	Function
*INSERT or data record with ID or record number	Insert records
*DELETE	Delete records
*REPLACE or data record with ID or record number	Replace records
*CHANGE	Change records
*END	End correction statements
*/ */	Delimit an entered record

### Format

Correction statements start with \*. The asterisk in column 1 is followed by the operation, i.e. the name of the correction statement, a space and the record designation. The record designation is called "recdes" in the description of the correction statement. The record designation is either a record number or a record identifier (see page 47).

**Record number** #number  
 "number" is a positive integer comprising up to 8 digits. The record number is the position of the member record relative to the start of the member (this numbering can be seen from the member listing by specifying PAR LST=mode/NUM). If "number" is greater than the highest record number in the member, the correction will be continued after the last record, i.e. records are appended to the member.

#### *Examples*

#26 denotes the 26th record in the input member.

#0 denotes the position before the first record of the member, i.e. a record is inserted before the first record of the member.

Record ID                      Letters, digits and special characters

The position and length of the ID (contents of the check field) are defined with the processing operand CHECK. This specification for record designation may therefore be used only when CHECK is not equal to NO. The length of the ID specified must be as defined in CHECK. Only leading zeros may be omitted. If it contains spaces, the ID must be delimited by the characters > and <.

If the ID does not occur in the input member, correction takes place

- before the first record,
- after the last record, or
- before the first record with a greater ID.

*Examples*

Check field in input member	ID
000315	315
xy z	>xy z<

In order to be able to address records with an empty check field in an unnumbered input member, it is also possible to use a mixture of record number and ID in a single correction run (or even in a single correction statement).

If the CHECK processing operand has been defined, data records can be included in the member to be corrected according to their check field (without having to use \*INSERT or \*REPLACE).

If there is a record with the specified check field, it is replaced by the record. If there is no record with the specified check field, the record is inserted before the first record with a higher check field. Records with an empty check field, or if CHECK=NO is set, are inserted at the current position.

Records following \*INS or \*REP must, unless immediately followed by \*INS, \*DEL, \*CHA, \*REP or \*END, be delimited by a record having the form \*/\*, so as to separate them from any subsequent data records with a record ID.

Description of the individual correction statements

\*INSERT Insert records

\*INSERT causes records to be inserted at desired positions. The records can either be input or simply copied from another member that is designated as the secondary member. The member to be corrected is considered to be the primary member.

Operation	Operands
*INS[ERT]	recdes[, [x=]member[(lib)][:recdes1[-recdes2]]]

- \*INSERT** Name of the correction statement.
- recdes** Records are inserted after the record having "recdes" as its record number or ID. If the specified record number or ID does not exist, records are inserted before the first record whose record number/ID is greater than "recdes". Those data records that are read subsequent to \*INSERT are inserted.
- x** Member type of secondary member:  
Allowed are S, M, J, P, D and X  
  
If no type is specified, the primary member type is assumed.
- member** Name of the secondary member.  
Multiple selection is not permitted.  
  
From the specified secondary member the records from "recdes1" through "recdes2" are inserted into the primary member.
- lib** Short library designation of the secondary member. Specification of the library can be omitted if the secondary member is contained in the input library of the primary member.
- recdes1** First or only record number or ID of a record from the secondary member that is to be inserted into the primary member.
- recdes2** All records from "recdes1" through "recdes2" are inserted from the secondary member into the primary member. If the recdes1 - recdes2 specification is omitted, the entire secondary member is inserted.

When records are transferred from a secondary member into the primary member, additional records can be specified after \*INSERT. They are inserted into the primary member following the records from the secondary member.

**\*DELETE** Delete records

\*DELETE deletes specified records or record ranges in a member.

Operation	Operands
*DEL[ETE]	recdes1[-recdes2]

**\*DELETE** Correction statement name.

recdes1-recdes2 Record range. All records having the addresses (record number or ID) from "recdes1" through "recdes2" are deleted. If only one record is to be deleted, the "-recdes2" specification can be omitted.

**\*REPLACE** Replace records

\*REPLACE causes records or record ranges in the member to be replaced by specified records.

Operation	Operands
*REP[LACE]	recdes1[-recdes2]

**\*REPLACE** Correction statement name.

recdes1-recdes2 Record range. All records having the addresses from "recdes1" through "recdes2" are replaced by the records following this correction statement. If only one record is to be replaced, the "-recdes2" specification can be omitted.

If there are more records following \*REPLACE than specified by recdes1 - recdes2, these records are nevertheless inserted at the current position.

\*CHANGE Change records

\*CHANGE is used to replace portions of text with a specific text or to insert text with a specific column alignment.

Operation	Operands
*CHA[NGE]	[recdes1[-recdes2]] 'text1' [<column>] [ { = : = } 'text2' ]

\*CHANGE Correction statement name.

recdes1-recdes2 Record range. In all records whose addresses (record numbers or check fields) are greater than or equal to "recdes1" and less than or equal to "recdes2", each search string found in the specified column range will be replaced by the correction text (replacement string).

text1 Search string. The string that is to be replaced is any desired sequence of characters, enclosed in apostrophes ( ' ') or double quotes ( " ").

Note the following with regard to double quotes ( " "):

In the record, only when the character immediately to the left of the first character in the string and the character immediately to the right of the last character in the string are not alphanumeric is the text considered to have been found.

If the blank character string " " is specified as the search string, the replacement string will be inserted commencing in the position defined by "column" (only one value must be specified). If the permissible record length (251 bytes) is exceeded, truncation takes place accordingly. Should characters (other than blanks) be lost as a result, an appropriate warning message will appear in the correction log.

column Defines a column range column1 - column2 in the record, within which the search string must begin. If only "column1" is specified, the search string must begin in this column in the record. If the search string comprises the blank character string " ", precisely one column value must be specified. This denotes the position at which the replacement string will be inserted. Specification of the column range may only be omitted if the search string does not consist of the blank character string. The entire record is then searched for the search string.



:=	Search string and replacement string must be of equal length.
=	Search string and replacement string may be of different lengths.
text2	Replacement string. Any desired string of characters, enclosed in apostrophes or quotes. The specified replacement string replaces the string found in the column range or (if the search string is the blank string) is inserted at the column position.

\*CHANGE statements for which no record range recdes1 - recdes2 is specified must always be located at the start of the correction statements for COR.

When a record range is specified, "recdes1" must be greater than or equal to a "recdes1" specified in a preceding correction statement (\*REP, \*DEL, \*INS). The value for "recdes2", however, is freely selectable.

**\*END** Terminate corrections

**\*END** terminates the correction inputs for COR.

Operation	Operands
<b>*END</b>	

**\*END** Correction statement name.

This correction statement does not have any operands.

*Example*

```
/SET-FILE-LINK FILE-NAME=SRC.LIB, LINK-NAME=LIB017
/START-PROGRAM $LMS
$LIB (17), BOTH
$CORS SRC82
**INSERT #3, SRC80:#3-#8
**DEL #21
**END
.
.
.
$END
```

Member SRC82 of library SRC.LIB is corrected. Following the record with record number #3, the records having the record numbers #3 through #8 are inserted from member SRC80. The record bearing the record ID #21 is deleted.

## CTL Define statement input source

CTL serves to define the input source for LMS statements.

Possible input sources are:

- the terminal
- system file SYSDTA or
- a library member.

Operation	Operands
CTL	$\left\{ \begin{array}{l} (\underline{\text{CMD}}) \\ [\text{SYS}]\text{DTA} \\ (\text{RDR}) \\ \text{member}[(\text{lib})] \\ ? \end{array} \right\}$

CTL	Statement name.
<u>CMD</u>	Statements are read from the terminal in interactive mode and from SYSDTA in batch mode.
SYSDTA	Statements are read from system file SYSDTA.
RDR	Has the same effect as the SYSDTA operand and is supported only for reasons of compatibility.
member	Designates the member from which the statements are read. Only one member may be specified, multiple selection is not permitted.  In the case of program libraries, the member must be type J; in the case of source libraries, type S. During this LMS run a source library may not be used for any other purpose.
lib	Short designation of the library.
?	The current value is logged.

### Example

```
/START-PROGRAM $LMS
$LIB ANWS,BOTH
$CTL BFL
$END
```

The statements from member BFL are executed. The last statement in this member is CTL (CMD). The statements, END in this case, are then read from the display terminal again.

## DEL Delete members

DEL deletes specified members of the assigned **input library**. The directory entries are thereby deleted and storage space is released.

In addition, members in program libraries are physically deleted, i.e. the data overwritten with binary zeros

- if the member contains a code for physical deletion
- if the processing operand DESTROY=YES has been set.

DEL is not permitted for sequential libraries.

Operation	Operands
DEL[x]	$\left\{ \begin{array}{l} \text{member} [ (\text{lib}) ] \\ (\text{lib}) \end{array} \right\} [ , \dots ]$

DELx	Statement name with specification of member type. All member types are permitted: S, M, R, J, P, C, D, X, H, L, F, U * Stands for all member types (only permissible for program libraries)
member	Member designation or multiple selection.
lib	Short designation of the input library. If no member is specified, the function applies to all members of the currently highest version of the referenced library, i.e. the whole library is deleted.

### Processing operands

TYPE	Defines the member type, if no type is specified in the statement itself.
DESTROY	Defines whether the members are to be physically deleted (only possible for program libraries).
REFERENCE	Defines which reference names the members to be deleted must have. If the reference condition is not satisfied, the member is not deleted. REFERENCE is permitted for member type R only. After a REFERENCE definition, R must be specified with DEL.

*Note*

Every update for a delta tree reorganizes the delta structure, i.e. data records which are no longer needed are deleted and unused storage space is released.

*Example*

```
/START-PROGRAM $LMS  
$LIB LS.LIB,BOTH  
$DELD DATA  
$DEL* TPROG  
$END
```

Member DATA of library LS.LIB is deleted. All members with the name TPROG in library LS.LIB are deleted, regardless of their member type.



## DUP Duplicate members and duplicate with structure

DUP permits the duplication of members or of entire delta trees, together with their original structure. Two formats are available for this purpose.

### Format 1: Duplicate members

Operation	Operands
DUP[x]	$\left\{ \begin{array}{l} \text{member, ... [ (lib) ] } \\ \text{(lib)} \end{array} \right\} [ > \text{memberu} ] [ , \{ \dots \} [ > \dots ] ]$ $[ , \text{BASEVERSION} = \left\{ \begin{array}{l} * \text{NONE} \\ \text{version} \\ * \text{HIGH} \end{array} \right\} ]$

### Format 2: Duplicate with structure

Operation	Operands
DUP[x]	name1 [ (lib) ] [ > name2 ] , STRUC=Y [ ES ]

**Format 1:** Duplicate members

DUP duplicates the specified members of the assigned input library, or an entire library, to the open output library. Duplicated members may be given new member designations in the process.

If the output library is a program library, the duplicated members may be stored either as non-delta or as delta members. If delta members are duplicated and the input library is identical with the output library, the duplicated delta members must be given new member names.

An output library must already have been assigned with LIB.

Certain record types may be excluded from the duplication process with the aid of processing operand STRIP in the case of R-type and C-type members.

Members can be selected for duplication by means of their reference names. If a reference condition is defined by means of the REFERENCE operand, only those members satisfying this condition will be duplicated.

Operation	Operands
DUP[x]	$\left\{ \begin{array}{l} \text{member}, \dots [(\text{lib})] \\ (\text{lib}) \end{array} \right\} [ > \text{memberu} ] [ , \{ \dots \} [ > \dots ] ]$ $[ , \text{BASEVERSION} = \left\{ \begin{array}{l} * \text{NONE} \\ \text{version} \\ * \text{HIGH} \end{array} \right\} ]$

**DUPx**

Statement name with specification of member type:

- For non-delta members, all member types are allowed:  
S, M, R, J, P, C, D, X, H, L, F, U
- For delta members, only the text-based member types are allowed: S, M, J, P, D
- \* Stands for all member types (only permitted for program libraries)

The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.

member	Member designation of the member to be duplicated, or multiple selection.  For "member", names which do not conform to LMS conventions are also permitted and such members can be further processed.
lib	Short designation of the input library.
memberu	Member designation of the output member, or construction specification.
BASEVERSION	Defines the base member for delta storage. It refers to all members to be generated (memberu). It must be the last operand of the statement. If this operand is not specified a non-delta member is created from a delta member.
=*NONE	"memberu" is stored as the first member (=base) of a delta tree (first generation).
=version	Base member is the member with "version" as the highest version designation. This member must be present and be stored as a delta member. A construction specification is not allowed.
=*HIGH	Base member is the member with the highest version designation (at the time of creating the delta member). The generated delta member is appended to this base member.

DUPx \*/\* may be used to duplicate all members of one type while retaining the member designations.

### Processing operands

TYPE	Defines the member type, if no type is specified in the statement itself.
OVERWRITE	Controls overwriting of identically named non-delta members in the output library. This processing operand is not allowed for delta members.
STRIP	Only for R-type and C-type members. Defines which records are to be excluded from the duplication process.
DESTROY	Defines whether a code for physical deletion is set in the output member (only possible for program libraries).



**REFERENCE** Defines which reference names the members to be duplicated must have. If the reference condition is not satisfied, the member is not duplicated.  
REFERENCE is permitted for member type R only. After a REFERENCE definition, R must be specified with DUP.

*Example*

```
/SET-FILE-LINK FILE-NAME=DUP.LIB, LINK-NAME=LIB002  
/START-PROGRAM $LMS  
$LIB OLD.LIB, IN  
$LIB (2), NEW, OUT  
$DUPS OLD1>DUP1/001/1983-02-07, OLD2>DUP2  
$END
```

Members from the library OLD.LIB are duplicated to a program library DUP.LIB that is to be created.

The link name LIB002 is assigned to the output library DUP.LIB that is to be created. The input library is assigned explicitly via LIB. Member OLD1 is duplicated as DUP1, version 1 with a new date. OLD2 is duplicated as DUP2 with unchanged version and unchanged date.

**Format 2:** Duplicate with structure

When this format is used, LMS recognizes the form in which members are stored in the PLAM libraries. Correspondingly, delta trees are duplicated as delta trees and all other members are written to the output file as non-delta members.

Operation	Operands
DUP[x]	name1 [ (lib) ] [>name2] , STRUC=Y[ES]

DUPx	Statement name with specification of member type. All member types are permitted: S, M, R, J, P, C, D, X, H, L, F, U  * Stands for all member types (only permissible for program libraries).
name1	Name (without version and date) of the input member, or multiple selection. Only one input member is allowed.
lib	Short designation of the input library.
name2	Name (without version and date) of the output member, or construction specification.
STRUC=YES	If "name1" specifies the name range of a delta tree, the delta tree is duplicated with its structure being retained. "name2" must not yet exist in the output library (processing operand OVERWRITE has no effect).  If "name1" specifies the name range of one or more non-delta members, all non-delta members with the name "name1" are duplicated as non-delta members (same as "name1/*"). If "name1" coincides with "name2", processing operand OVERWRITE takes effect.

*Notes*

- If the duplication process is aborted, the copied part of a delta tree is retained.
- No version and date specifications are allowed for "name1" and "name2".

**Processing operands**

TYPE	Defines the member type, if no type is specified in the statement itself.
DESTROY	Defines whether a code for physical deletion is set in the output member (only possible for program libraries).



## EDT/EDR      Create, correct and view text members and files

EDT/EDR invokes the file editing programs EDT and EDOR respectively, for the creation, correction or display of either text members or files. EDT/EDR consequently has three different formats.

### Format 1:      Create and correct text members

Operation	Operands
EDT [x]	member [ (lib) ] [>memberu]
EDR [x]	member [ (lib) ] [>memberu]

### Format 2:      Create and correct files

Operation	Operands
EDT	
EDR	

### Format 3:      View members

Operation	Operands
EDT [x]	memberu [ (lib) ] >*DUMMY
EDR [x]	memberu [ (lib) ] >*DUMMY

#### *Note*

PAM files stored in the library cannot be processed using EDT/EDR.

**Format 1:** Creation and correction of text members

This EDT/EDR format invokes EDT or EDOR and reads the specified member from the assigned input library. When EDT/EDOR is terminated, the corrected member with any new name assigned is written to the assigned output library.

The editors are described in the manuals on EDT [5] and EDOR [4].

LMS supports EDT versions higher than V16.2A.

In the case of sequential libraries, the input and output libraries must not be identical. Where other library types are concerned, input and output libraries may be identical.

Operation	Operands
EDT[x]	member[ (lib) ] [>memberu]
EDR[x]	member[ (lib) ] [>memberu]

EDTx }  
EDRx }

Statement names, with specification of member type:  
S, M, P, J, D, X

The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.

- member Member designation of the member to be corrected or generated, or multiple selection.
- lib Short designation of the input library.
- memberu Member designation of the output member or construction specification.

**Processing operands**

- TYPE Defines the member type if no type is specified in the statement itself.
- OVERWRITE Controls overwriting of identically named members in the output library. If  
input library = output library and member = memberu,  
this processing operand will have no effect and the input member is overwritten.

DESTROY	Defines whether a code for physical deletion is entered in the output member and whether any scratch file which may have been created is physically deleted after use (only possible in conjunction with program libraries).
CHECK	Defines the position and length of the check field in input records and checks for ascending numbering.
RANGE	Defines the position and length of the check field in output records.
STRING	Specifies whether the ISAM key is to be stored in the check field, or specifies the character string to be entered left-justified in the check field of output records.
VALUE	Specifies the initial value and increment for renumbering in the check field.

### Scratch file

LMS generates the following scratch file under certain conditions when EDT is called, and always when EDOR is called:

```
S.LMS.TSNnnnn.date.time-of-day.member
```

"member" can be up to 9 characters long. Member names exceeding this limit are truncated after the first 9 characters. If the "member" suffix would result in an illegal BS2000 file name (e.g. '.' as the ninth character), LMS forms a scratch file name without the member designation:

```
S.LMS.TSNnnnn.date.time-of-day
```

If RECORD-FORMAT=FIXED is stored in the input member, the function is aborted with an error message. The same applies to KEY-POSITION>5.

If ISAM keys are stored in the input member, they are first transferred to the scratch file and then (following correction) to the output member.

### Transfer of ISAM key from check field

Processing operand CHECK=start/length defines a check field in the input records. The contents of this check field are entered left-justified in the ISAM key of the record. If check field (length) and ISAM key (KEY-LENGTH) have varying lengths, LMS takes the following action:

KEY-LENGTH<length: the check field is truncated on the right.

KEY-LENGTH>length: the ISAM key is padded with zeros on the right.

If no ISAM keys have been stored in the input member, an ISAM file with KEY-POSITION=5 and KEY-LENGTH=8 is created. By default, LMS then generates ISAM keys with an initial value of 1000 and an increment of 1000. If the member is too large for this increment (more than 100,000 records), the increment is calculated from the number of records.

The transfer is effected by means of an internal SEL call. This call permits the same functions as those provided by SEL with respect to the ISAM keys (see the CHECK, STRING, VALUE and RANGE processing operands).

### Editor run

- EDTx membername:  
LMS passes the member records on to EDT. The member is then available in virtual memory. The line number displayed by EDT gives the first six digits of the ISAM keys.  
  
A scratch file with linkname=EDTISAM will be generated if processing operands STRING and CHECK are used or when the stored ISAM keys specify a KEY-POSITION less than 5 or KEY-LENGTH less than 8.
- If a file was assigned the link name EDRPRIMR outside of LMS and this assignment still applies when the user wants to process a member via EDRx, the assignment must first be cancelled by means of /REMOVE-FILE-LINK LINK-NAME=EDRPRIMR, because otherwise the file assigned with EDRPRIMR will be processed. On EDOR termination, however, only the scratch file with the member data will be written back.
- Old libraries (OSM) are closed. Other users can access the library and the member. Program libraries are not closed, the member processed is locked to other users.
- The appropriate editor is now invoked as a subroutine. If a scratch file has been generated, it is opened and may be processed.
- When the editor is invoked LMS-STXIT is terminated, and when control returns from the editor it is started again.

- Termination of the editors:

EDT:

RETURN from work file 0:

The current work file is added as a member to the output library. The EDT data (files in virtual memory, variables,...) remains intact. This data is released only in the case of a severe EDT error.

HALT from work file 0:

```
LMS0420: EDITED ELEMENT
(type)membername/version[(variantnumber)]/date TO BE ADDED ?
REPLY (Y=YES, N=NO)?
```

The response determines whether or not the current work file is added as a member. The EDT data remains intact. This data is released only in case of a severe EDT error.

HALT/RETURN from word file  $\neq$  0:

The following message is issued:

```
LMS0420: EDITED ELEMENT
(type)membername/version[(variantnumber)]/date TO BE ADDED ?
REPLY (Y=YES, N=NO)?
```

If 'N' is entered, the member is not added. If the reply is 'Y', the following dialog is then conducted with the user:

```
LMS0421: WORKFILE TO BE ADDED (0 = WORKFILE(0),..., N = NONE) If
the reply is 'N', LMS returns to the EDT work file currently being
processed.
```

EDT in batch mode:

@RETURN:

The corrected member is added from work file 0 to the output library, provided that the output library is not empty.

@HALT:

The corrected member is not added to the output library.

EDOR:

H!H The corrected member is added to the output library.



**Adding processed members to the output library**

The following applies, depending on the file processing program used:

- A member processed by EDOR is added to the library only if
  - the scratch file is not empty
  - EDOR is terminated with H!H
  
- A member processed by EDT is added to the library only if
  - EDT is terminated with RETURN from work file 0 and the scratch file is not empty
  - EDT is terminated with RETURN from a work file ≠ or HALT and the response to the subsequent message "EDITED ELEMENT ..." is "Y"
  - EDT is terminated with RETURN in batch mode and work file 0 is not empty.



**Format 2:** Creation and correction of files

This EDT/EDR format simply invokes EDT/EDOR. EDT and/or EDOR are then available for file processing under the conditions described in the relevant manuals [4], [5]. Once EDT/EDOR is terminated, the interrupted LMS run is resumed. The EDT data (files in virtual memory, variables,...) is retained.

Operation	Operands
EDT	
EDR	

**Format 3:** Display of members

This format enables the user to view members without assigning an output library.

Operation	Operands
EDT[x]	memberu[ (lib) ]>*DUMMY
EDR[x]	memberu[ (lib) ]>*DUMMY

memberu	Member designation of the input member. Multiple selection is allowed in restricted form (no list input) for program libraries.
lib	Short designation of the input library.
*DUMMY	The member readied for the editor is not written back; in addition the following applies: <ul style="list-style-type: none"> <li>– an output library need not be assigned;</li> <li>– version and date entries are checked for syntax errors;</li> <li>– memberu must exist;</li> <li>– any scratch file will be erased.</li> </ul>

## END Terminate LMS run

END concludes the LMS program. All libraries that are still open are closed.

In interactive mode, LMS always terminates in a normal manner; in batch mode, or in procedures, the type of termination depends on the termination code (see TERMINATE processing operand).

If an internal LMS termination code is set, then this is output simultaneously with the LMS termination message (LMS-TERM-MSG).

Operation	Operands
END	

The completion code indicates the most serious error that has occurred. It is stored in the monitoring job variable specified in the call

```
/START-PROGRAM LMS,MONJV=<name>
```

Completion code	Meaning
0	No error
1	Warning messages were output
2	Error with interrupt code occurred (see processing operand TERMINATE)
3	Internal LMS error (with dump)

Once LMS is terminated, the status indicator of the program monitoring job variable contains the completion code (see above) in the first byte, and the internal termination code (see PAR TERM) in the fourth byte.

## LIB Assign and close libraries

LIB permits input and output libraries to be created, opened and closed. LMS reads members from the input library and outputs members to the output library.

DEL and NAM are exceptions to the above, since they only affect the assigned input library.

LIB has three formats:

### Format 1: Assign libraries

Operation	Operands
LIB	$\left\{ \begin{array}{l} \text{FILE=libname} \\ \text{LINK=linkname} \\ \text{[LIBRARY=]name} \\ \text{[LID=] (lib)} \end{array} \right\} [, [\text{USAGE=}] \left\{ \begin{array}{l} \text{IN} \\ \text{OUT} \\ \text{BOTH} \end{array} \right\}]$ $[, [\text{FORMAT=}] \left\{ \begin{array}{l} \text{PL} \\ \text{OML} \\ \text{OSM} \end{array} \right\}] [, [\text{STATE=}] \left\{ \begin{array}{l} \text{O[LD]} \\ \text{N[EW]} \\ \text{A[NY]} \end{array} \right\}]$

### Format 2: Close libraries

Operation	Operands
LIB	$\text{C[LOSE] } [, \left\{ \begin{array}{l} \text{FILE=libname} \\ \text{LINK=linkname} \\ \text{[LIBRARY=]name} \\ \text{[LID=] (lib)} \end{array} \right\}]$

### Format 3: Display the assigned libraries

Operation	Operands
LIB	?

**Format 1:** Assign libraries

LIB assigns libraries. It permits specifications to be made in order to

- define the library as an input or output library, or both
- define the type of library (program library, source library, macro library or object module library)
- define whether the library is to be newly created, whether it already exists, or whether it is to be newly created if required.

LIB closes the input library previously assigned with LIB by assigning a new input library. The same applies to output libraries.

The newly assigned library is opened.

Operation	Operands
LIB	$\left. \begin{array}{l} \text{FILE=libname} \\ \text{LINK=linkname} \\ \text{[LIBRARY=]name} \\ \text{[LID=] (lib)} \end{array} \right\} [ , [\text{USAGE=}] \left\{ \begin{array}{l} \text{IN} \\ \text{OUT} \\ \text{BOTH} \end{array} \right\} ]$ $[ , [\text{FORMAT=}] \left\{ \begin{array}{l} \text{PL} \\ \text{OML} \\ \text{OSM} \end{array} \right\} ] [ , [\text{STATE=}] \left\{ \begin{array}{l} \text{O[LD]} \\ \text{N[EW]} \\ \text{A[NY]} \end{array} \right\} ]$

- LIB Statement name.
- FILE=libname Fully qualified file name of the library. "pathname" may also be specified for "libname" (see the manual "User Commands (SDF Format)" [7]).
- LINK=linkname Specifies the link name assigned to the library in a /SET-FILE-LINK command.
- LIBRARY=name LMS first attempts to interpret "name" as a link name. If no such link name has been assigned previously with a /SET-FILE-LINK command, "name" will be interpreted as a library name.
- The keyword "LIBRARY=" may be omitted. In this case, the library name must not be "CLOSE", since otherwise LIB format 2 will be assumed.

LID=(lib)	Specifies the short library designation consisting of up to 3 digits. The short library designation must have been defined previously using a /SET-FILE-LINK command with the link name LIBlib.
USAGE	This operand specifies whether the library is to be used for input, for output or for both.
=IN	During the LMS run, the library is the input library. Default value when STATE=OLD.
=OUT	During the LMS run, the library is the output library. Default value when STATE=NEW or STATE=ANY.
=BOTH	During the LMS run, the library is both the input and the output library.
FORMAT	This operand specifies the type of the library to be assigned.  The FORMAT operand is only necessary when new libraries are to be created and the default value PL is undesirable. The library type of existing libraries is identified by LMS.
=PL	Program Library.
=OML	Object Module Library.
=OSM	Old Source/Macro library.
STATE	Determines whether the library is to be created, whether it already exists, or whether it is to be created if required.
=O[LD]	The library already exists.
=N[EW]	The library is to be created. If the library has already been created, the statement is rejected and an error message is issued.
=A[NY]	The library is created if it does not yet exist.

The short library designation in DEL, NAM, NUM, COR, UPD, EDT, EDR, DUP, COM, LST and TOC permits a library other than the standard input library to be assigned. This input library is only valid while the statement is executing. Afterwards, the standard input assignment will resume its validity.

*Example*

```
/SET-FILE-LINK FILE-NAME=LMS.BEI, LINK-NAME=BEILINK
/SET-FILE-LINK FILE-NAME=LMS.EINB, LINK-NAME=LIB001
/START-PROGRAM $LMS
$LIB FILE=LMS.AUS, OUT, NEW
$LIB LINK=BEILINK, IN,
.
.
.
$LIB LID=(001), USAGE=IN
.
.
.
$END
```

The library LMS.BEI is assigned with a /SET-FILE-LINK command, and linked to LMS as the input library using LIB with the link name BEILINK and the operand value IN.

The output library (OUT,NEW) LMS.AUS to be created is specified in LIB via the file name.

On input of the third LIB, which assigns the library LMS.EINB as the input library by way of the short library designation, the library LMS.BEI is closed.

**Assignment of sequential libraries**

LIB permits sequential libraries to be assigned as input libraries only. In order to create sequential libraries LIBOUT (lib),NEWLIB (see page 330) must be used.

Existing sequential output libraries are assigned with LIBOUT (lib).

**Format 2:** Close libraries

This format of LIB is used to close libraries. The input/output library assignment is cancelled.

Operation	Operands
LIB	$C[LOSE] \left[ \begin{array}{l} FILE=libname \\ LINK=linkname \\ [LIBRARY=]name \\ [LID=] (lib) \end{array} \right]$

LIB	Statement name.
C[LOSE]	Causes the library, or libraries, to be closed.
FILE=libname	Specifies the fully qualified file name of the library to be closed. "pathname" may also be specified for "libname" (see the manual "User Commands (SDF Format)" [7]).
LINK=linkname	Specifies the link name of the library to be closed; this library was assigned with a /SET-FILE-LINK command and must be known to LMS.
LIBRARY=name	LMS first attempts to interpret "name" as the assigned link name of the library to be closed. If no such link name is found, "name" is interpreted as a library name. The link name or the library name must be known to LMS.
LID=(lib)	Specifies the short designation of the library to be closed. The library must have been assigned with the /SET-FILE-LINK command via the link name LIBlib, and must be known to LMS.

*Example*

```

/START-PROGRAM $LMS
$LIB LMS.TEST, BOTH
.
.
.
$LIB C, LMS.TEST
.
.
.
$END

```

The library LMS.TEST is assigned as both the input and the output library with the first LIB, and closed with the second LIB.



**Format 3:** Display assigned libraries

This format of LIB provides information on the libraries used during the LMS run.

The following information is output:

- library function (input library or output library, or both)
- library status (opened or closed)
- library format
- assigned short designation, if any
- assigned link name, if any
- library file names

Operation	Operands
LIB	?

## LST List members

LST lists the specified members or all the members of the assigned input library.

Depending on the value of PRT, the output medium is

- the terminal, and/or
- system file SYSLST, or
- a library member.

The scope and format of the listing is controlled by processing operands CSECT, FORMAT, LINE, PATH, SLICE, INFO and NEWFORM.

For the listing of members of type S, M, J, D, P or X, the user may also branch to user routines (see the USE statement).

Operation	Operands
LST[x]	$\left\{ \begin{array}{l} \text{member [ (lib) ]} \\ \text{(lib)} \end{array} \right\} [ , \dots ]$

LSTx	<p>Statement name with specification of member type: All member types are permitted: S, M, R, J, P, C, D, X, H, L, F, U</p> <p>* Stands for all member types (only permitted for program libraries)</p> <p>The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.</p>
member	Designates the members to be listed, or multiple selection.
lib	Short designation of the input library.

Since list members (type P in program libraries and type S in source libraries) may only be output to the SYSLST system file, PRT (LST) must also be specified. In this case the processing operand LINE is ignored. (In the case of list members in source libraries, processing operand PAR FORMAT=P is additionally required.)

LMS error messages are always output to SYSOUT, even if PRT (LST) has been set.

**Processing operands**

CSECT	Defines which control section is to be listed, for type L.
TYPE	Defines the member type if no type is specified in the statement itself.
FORMAT	Determines whether records are output in character form, in hexadecimal form, or using a combination of these. For more details see the "Systems Standards" manual [12].
INFO	Determines whether all records, only certain record types, certain areas or only the most import member data are output.
LINE	Defines the number of lines and columns on a log page for output to system file SYSLST or to a member.
PATH	Defines the sub-LLM to be listed, for type L.
REFERENCE	Defines which reference names the members to be listed must have. If the reference condition is not satisfied, the member is not included in the list. REFERENCE is permitted for member type R only. After a REFERENCE definition, R must be specified with LST.
SLICE	Defines the member slice to be listed, for type L.
BASE	Defines the base address for relations between address areas (see processing operand INFO); for BS2000 load modules only.
SEGMENT	Defines the segments to be listed (for BS2000 load modules only).
NEWFORM	Controls line feed of the LMS log.
LST	Supported for reasons of compatibility only. The operand values are replaced internally by the values of the processing operands FORMAT and INFO.

*Example*

```
/SET-FILE-LINK FILE-NAME=TEST.LIB, LINK-NAME=LIB003  
/START-PROGRAM $LMS  
$PAR INFO=SUMMARY  
$LSTS (3)  
$END
```

Information on all S-type members of program library TEST.LIB is listed.

## NAM Rename members

NAM renames the specified members of the assigned input library. The name is changed solely in the input library's directory.

NAM is not permitted for delta members and sequential libraries.

Operation	Operands
NAM[x]	$\left\{ \begin{array}{l} \text{member}, \dots [(\text{lib})] \\ (\text{lib}) \end{array} \right\} > \text{memberu}[, \{ \dots \} > \dots ]$

NAMx	Statement name with specification of member type. All member types are permitted: S, M, R, J, P, C, D, X, H, L, F, U  * Stands for all member types (only permitted for program libraries)  The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Member designation that is to be changed, or multiple selection.  For "member" names that do not conform to LMS conventions are also permitted so as to enable the further processing of such members.
memberu	New member designation; construction specification also permitted.
lib	Short designation of the input library.

### Processing operands

TYPE	Defines the member type, if no type is defined in the statement itself.
DESTROY	Determines whether a code for physical deletion is set in the renamed members (only possible in conjunction with program libraries).
OVERWRITE	Controls overwriting of identically named members.

*Example*

```
/START-PROGRAM $LMS  
$LIB PROD.LIB, BOTH  
$NAMM MAXOUT>MACOUT  
$END
```

The macro MAXOUT is renamed MACOUT.

## NOP Dummy function

NOP causes no action in LMS. In procedures it can be used to reserve space.

Operation	Operands
NOP	[string]

NOP                    Statement name.

string                 Any text.

### *Note*

"!" and X'15' (NEW-LINE) must not occur in text, as they are interpreted as statement delimiters.

### *Example*

NOP is used in a procedure to reserve space for a statement.

```

/PROC C, (&ANW=NOP, ...), SUBDTA=&
.
.
.
/MODIFY-JOB-SWITCHES ON=(1)
/START-PROGRAM $LMS
.
.
.
&ANW ELEM(1)
.
.
.
END
/MODIFY-JOB-SWITCHES OFF=(1)
.
.
.
/END-PROCEDURE

```

The procedure is called by the command /CALL-PROCEDURE ..., ANW=LSTS. The procedure then causes the member ELEM from the library with short designation (1) to be output.

## NUM Number member records

NUM numbers the check field of the specified members. The generated member is transferred to the assigned output library.

The position, length and contents of the check field are defined by processing operands.

The record IDs are provided for record identification; they may be checked for ascending sequence in further statements. For further information concerning check fields, see page 48.

Input and output libraries must not be identical when sequential libraries are involved. Where other libraries are concerned, input and output libraries may be identical.

Operation	Operands
NUM[x]	$\left. \begin{array}{l} \{ \text{member} [ (\text{lib}) ] \\ \{ (\text{lib}) \} \end{array} \right\} [ > \text{memberu} ]$

NUMx	Statement name with specification of the member type: S, M, J, P, D, X
	The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Member designation or multiple selection.
lib	Short designation of the input library.
memberu	Member designation of the output member, or construction specification.

### Processing operands

TYPE	Defines the member type if no type is defined in the statement itself.
DESTROY	Determines whether a code is set in the output member for physical deletion (only possible in conjunction with program libraries).
RANGE	Defines the position and length of the check field in output records.
VALUE	Specifies the initial value and increment for numbering in the check field.

STRING	Specifies the string to be entered left-justified in the check field of output records.
OVERWRITE	Controls overwriting of identically named members in output library. If <pre>input library = output library and member = memberu,</pre> this processing operand will have no effect and the input member is overwritten.

The member is numbered in accordance with the current values of the processing operands RANGE, VALUE and STRING.

*Example*

```
/START-PROGRAM $LMS  
$LIB PROD1.LIB,BOTH  
$PAR RANGE=1/8,VALUE=10000000/1000  
$NUMS ELM2*  
$END
```

The members whose names start with ELM2 are numbered. The check field of the first record receives the value 10000000; the second record 10001000, etc.



## PAR Set processing operands

PAR is used to set processing operands. A processing operand can both define a processing mode and specify the values required for processing.

Operation	Operands
PAR	$\left[ \left\{ \begin{array}{l} \text{parname} = \{ \text{parvalue} \} \\ ? \end{array} \right\} , \{ \dots \} \right]$

PAR	Statement name.
parname	Name of the processing operand (see page 203 ff).
parvalue	Value of the processing operand (see page 203 ff).
?	The current values are listed for all the processing operands, or for the processing operands specified as "parname".

Default values are defined for all the processing operands. At the start of an LMS run all the processing operands are set to these default values. The default values are given under the descriptions of the individual operands (see page 209 ff). If, in PAR, the values are not specified or are specified but contain errors, the appropriate default values are set. The processing operands may be specified in any order. Where multiple specification of the same operand occurs, the last instance will always be the one to apply. The specifications remain valid until a new value is explicitly defined or until a PAR without operands causes the operands to be reset to the default values. The descriptions of the individual functions indicate which processing operands affect which function.

See page 205 for a summary of the processing operands.

## PRT Control log output

PRT defines the output medium for LMS logs.

The output medium may be:

- the terminal
- the system file SYSLST, or
- a library member.

If the log is written to a member, LMS will generate a P-type member for program libraries, and an S-type member for source libraries. If the library to which the member is written is a source library, it must not be the standard input or output library for the current LMS run.

Operation	Operands
PRT	$\left\{ \begin{array}{l} (LST) \\ ([SYS]OUT) \\ (CON) \\ (BOTH) \\ member[ (lib) ] \\ ? \end{array} \right\}$

PRT	Statement name.
LST	Outputs the log to system file SYSLST.
SYSOUT	Outputs the log to system file SYSOUT (i.e. to the data display terminal in interactive mode).
CON	Has the same effect as the SYSOUT operand, and is supported for reasons of compatibility only.
BOTH	Outputs the log both to the terminal and to system file SYSLST.
member	Outputs the log to library member "member".
lib	Short designation of the library to which the member is written.
?	The current value is logged.

*Note*

LMS error messages are always output to SYSOUT, even if PRT(LST) has been set.

**Processing operands**

LINE	Controls the number of lines and columns of a log page (only meaningful for SYSLST and "member").
LOG	Controls the scope of the log; in order to receive a log, PAR LOG=MED or PAR LOG=MAX must have been set (default value LOG=MIN).
NEWFORM	Defines feed control for logs.
OVERWRITE	Controls overwriting of identically named members in the output library.
DESTROY	Determines whether a code for physical deletion is to be set in the log (only possible in conjunction with program libraries).

*Example*

```
/SET-FILE-LINK FILE-NAME=PROT.LIB, LINK-NAME=LIB002
/START-PROGRAM $LMS
$PRT PROELEM(2)
$LIB EINAUS.LIB, BOTH
:
:
$END
```

The log for this LMS run is output to member PROELEM of library PROT.LIB.

## RST Restart after test mode

RST causes run mode to be resumed following previous activation of test mode on account of an error (see page 70). The previous input/output libraries are closed. After that they are "undefined" and must be redefined.

Operation	Operands
RST	[STOP]

RST                    Statement name.

STOP                    Run mode is resumed.  
 However, the internal termination code is not reset.

If the termination bit is set when LMS terminates, LMS terminates with TERMJ instead of TERM.  
 LMS issues the end message "TERMINATED".

If this operand is omitted, the termination code (if set) will be reset and run mode resumed.

### Processing operand

TEST                    Activates test mode.

If TEST=YES is set, RST has no effect.

## SEL Output library members to files and FMS libraries

SEL outputs specified members of the assigned input libraries to files or FMS libraries. SEL has two formats for performing these different functions.

**Format 1:** Output members to files

Operation	Operands
SEL[x]	member, ... [>{ [prefix.] (name) [.suffix] filename LINK=linkname }]

**Format 2:** Output members to FMS libraries

Operation	Operands
SEL[x]	member>FMS=fmslib ( fmsmember )

**Format 1:** Output members to files

This format of SEL outputs library members to files.

LMS creates files in accordance with

- the file attributes stored (PAR KEY=YES) and the FCBTYPE processing operand
- the entry in the task file table (TFT), if the file has been assigned via the link name. This specification has priority over the file attributes which have been stored.
- the catalog entry.

The files can have RECORD-FORMAT=UNDEFINED and arbitrary BUFFER-LENGTH and RECORD-SIZE values. However, the maximum record length of 32 Kbytes (including the record header) must not be exceeded.

If the ISAM keys of an ISAM file have been included in the member (by means of PAR KEY=YES), the ISAM keys are also output when SEL is issued. In this case the processing operands CHECK, STRING and VALUE are ignored.

For C-type members and for PAM files under type X, PAM files are again generated when selection is performed.

Operation	Operands
SEL[x]	member, ... [ $\left\{ \begin{array}{l} [\text{prefix.}] (\text{name}) [.\text{suffix}] \\ \text{filename} \\ \text{LINK=linkname} \end{array} \right\}$ ]

SELx	Statement name with specification of member type: S, M, R, J, P, D, X  C is permitted only for BS2000 load modules.  The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Member designation of the member to be output. Multiple selection is permitted if <ul style="list-style-type: none"> <li>– filename is not specified</li> <li>– a construction specification was made for "name" in the [prefix.](name)[.suffix] expression.</li> </ul>

filename	Fully qualified file name of the file to be generated. Multiple selection is not allowed. "pathname" may also be specified for "filename" (see the manual "User Commands (SDF Format)" [7]).
LINK=linkname	Link name referencing a file which has been assigned with the /SET-FILE-LINK command. The assigned file must not be a tape file, rather it must be a disk file.
[prefix.](name)[.suffix]	If multiple selection is specified for the member designation "member", several members may be output to different files using this expression. "name" must be a construction specification in this case.
prefix.	Specifies the common prefixed portion of names of files to be created. "prefix" must end with a period.
.suffix	Specifies the common suffixed portion of names of files to be created. "suffix" must begin with a period.
name	Specifies part of a name by which prefix and/or suffix are supplemented to form one or more fully qualified file names. If a construction specification is given for "name", this part of the file name is formed from the member names.

*Note*

Valid member names are not always permitted as file names.

**Processing operands**

TYPE	Defines the member type if no type is specified in the statement itself.
FCBTYPE	Defines the FCB type of the output file for members which contain text and in accordance with the file attributes specified with PAR KEY.
OVERWRITE	Determines whether an identically named file is overwritten, not overwritten, or expanded by the records of the input member.
CHECK	Defines a check field in input records.
STRING	Specifies a character string to be entered left-justified in the ISAM key field.

VALUE Defines the initial value and increment of the numeric value to be entered in the ISAM key.

PHASE Determines whether an NK or a PK phase is generated.

Processing operands CHECK, STRING and VALUE are ignored if the ISAM key is stored in the member.

### Generation of ISAM files

When members are output to ISAM files, LMS generates the ISAM keys as follows:

- If the ISAM keys are also added (PAR KEY=YES) when an ISAM file is included as a library member, LMS generates the ISAM file with those ISAM keys which have been stored.
- If no ISAM keys have been stored in the member, LMS will generate them.

1. Generation of ISAM key by means of VALUE and STRING processing operands

VALUE defines the initial value and increment for the right-justified numeric value in the ISAM key. Any character string defined by STRING is entered left-justified in the ISAM key field.

The CHECK processing operand must have the value NO in this case, i.e. no check field may be defined in the input records.

2. Generation of ISAM key by way of default values

If processing operands CHECK and VALUE have the value NO, LMS will generate the ISAM key with an initial value of 1000 and an increment of 1000, as a standard procedure. The generated values are entered right-justified in the ISAM key. If processing operand STRING has defined a character string, this string will be entered left-justified in the key field.

3. Transfer of ISAM key from check field

Processing operand CHECK=start/length defines a check field in the input records. The contents of this check field are entered left-justified in the ISAM key of the record. If check field (length) and ISAM key (KEY-LENGTH) have varying lengths, LMS takes the following action:

KEY-LENGTH<length: the check field is truncated on the right.

KEY-LENGTH>length: the ISAM key is padded with zeros on the right.

Processing operands STRING and VALUE are not interpreted here.



*Notes*

- R-type members are output up to the END record. Any records which come afterwards are ignored.
- Correction journal records (TXTP) are not included in the output in the case of C-type members.
- RECORD-SIZE is supplied with values only when RECORD-FORMAT=FIXED; when RECORD-FORMAT=VARIABLE, the value is 0.
- If, in the case of SEL and PAR OVERWRITE=YES, a file with the name of the file to be created already exists, the existing cataloged file must have the same attributes as the file to be generated.

*Example 1*

```
/SET-FILE-LINK FILE-NAME=AUSDAT, LINK-NAME=AUSD, -  
                ACCESS-METHOD=SAM, RECORD-FORMAT=VARIABLE  
/START-PROGRAM $LMS  
$LIB EIN.BIB, IN  
$SELS ELEM1>LINK=AUSD  
$END
```

SEL causes member ELEM1 to be output to file AUSDAT, which has the specified file attributes.

*Example 2*

If all members of a library are to be output under their respective names, the following statement should be entered:

```
$SEL[x] *
```

**Format 2:** Output members to FMS libraries

This format of SEL outputs a member from an LMS library to an FMS library. The member is written to the FMS library with the attributes defined in the LMS library. A SAM file is generated in the FMS library as a standard procedure. If a member is overwritten, it will receive the existing attributes.

This statement causes the FMS program (see the "FMS" manual [10]) to be invoked as a subroutine.

Operation	Operands
SEL[x]	member>FMS=fmslib (fmsmember)

SELx	Statement name with specification of member type: S, M, R, J, P, D, X  The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Designation of the member to be output. Multiple selection is not permitted.
fmslib	Fully qualified file name of the FMS library. "pathname" can also be specified for "fmslib" (see the manual "User Commands (SDF Format)" [7]).
fmsmember	Complete designation of the member generated in the FMS library.

**Processing operands**

CHECK	Defines a check field in input records.
OVERWRITE	Controls overwriting of identically named FMS members. Only the values YES, NO or ONLY may be specified.
STRING	Specifies a character string to be entered left-justified in the ISAM key field.
VALUE	Defines the initial value and increment of the numeric value to be entered in the ISAM key.

*Example*

```
/START-PROGRAM $LMS  
$LIB ERT.LIB,BOTH  
$SELS ERTEL>FMS=FMSERT(ERT3)  
$DELS ERTEL  
$END
```

Member ERTEL is written as member ERT3 to the FMS library and is subsequently deleted in library ERT.LIB.

## SUM Store comparison statistics

Comparison statistics are generated when members are compared with the aid of COM. COM stores these comparison statistics in a sum field with the designation S1, provided processing operand SUM is set before COM is issued.

SUM performs the following actions:

- SUMPRT S1      Outputs the summed comparison statistics in sum field S1, together with a title.
- SUMADD S1>S2    Adds the comparison statistics stored in sum field S1 to sum field S2.
- SUMDEL S1      Sum field S1 is deleted.

Operation	Operands
SUM	[ 'text' ]

SUM              Statement name.

text             Text that is output as a title prior to the comparison statistics.

Apostrophes within the text must be given in duplicate. Default value: AREA S1

### Processing operand

SUM              Is defined prior to COM and controls storage of the comparison statistics.

## SUMPRT Output comparison statistics

SUMPRT serves to output one of the sum fields used, S1 or S2. The text enclosed in apostrophes is output as a title.

Operation	Operands
SUMPRT	$\left\{ \begin{array}{l} S1 \\ S2 \end{array} \right\} [, 'text']$

SUMPRT	Statement name.
S1	Designation of the sum field to be output.
S2	
text	Title for output of the sum field. Apostrophes within the text must be specified in duplicate.

## SUMADD Add comparison statistics

SUMADD adds the comparison statistics stored in sum field S1 or S2 to sum field S2 or S1, respectively.

sum field S2 = sum field S2 + sum field S1 (SUMADD S1>S2)

Operation	Operands
SUMADD	$\left\{ \begin{array}{l} S1 \\ S2 \end{array} \right\} > \left\{ \begin{array}{l} S1 \\ S2 \end{array} \right\}$

SUMADD	Statement name.
--------	-----------------

## SUMDEL Delete comparison statistics

SUMDEL deletes the comparison statistics in the specified sum field.

Operation	Operands
SUMDEL	{ S1 } { S2 }

SUMDEL Statement name.

S1 Designation of the sum field to be deleted.

S2

## SYS Issue system commands

SYS enables system commands that are processed with the CMD macro to be issued without leaving program mode. If no operand is specified, SYS has the same effect as a BREAK command.

A return to program mode is effected by means of the /RESUME-PROGRAM command.

Operation	Operand
SYS	[ { 'systemcommand' } ] [ systemcommand ]

SYS Statement name.

systemcommand Name of a command with the necessary or desired operands. The system command is passed unchanged to the command processor.

## TCH Change terminal characteristics

TCH controls screen overflow and specifies whether the output is to appear on a new screen or in roll-up mode.

Operation	Operands
TCH	$\left\{ \left[ \begin{array}{l} \text{N[O]} \\ \text{A[CK]} \\ \text{T[TIMER]} \\ \text{?} \end{array} \right] \left[ \text{, N[NEWSCREEN]} = \left\{ \begin{array}{l} \text{Y[ES]} \\ \text{N[O]} \\ \text{?} \end{array} \right\} \right] \right\}$

**OFLOW** This operand is based on the OVERFLOW-CONTROL operand of the /MODIFY-TERMINAL-OPTIONS command (see the manual "User Commands (SDF Format)" [7]). It controls overwriting of a full screen of data (overflow control).

The default value for OFLOW is that which has been specified by the user or the system administrator in the /MODIFY-TERMINAL-OPTIONS command.

**=NO** No overflow control, i.e. when the screen is full and further data is waiting to be displayed, the screen is overwritten.

**=ACK** When the screen is full and further data is waiting to be displayed, LMS issues the following message:

```
PLEASE ACKNOWLEDGE (NO/TIMER/<ANY INPUT>) /
INTERRUPT (NE/NS/NI) :
```

The screen is not overwritten until the user responds with one of the entries prompted in the PLEASE ACKNOWLEDGE message. Input options NE/NS/NI are described in the section on "Control of screen overflow" (see page 69).

<ANY INPUT> means that any option other than those prompted will cause the next screen to be displayed.

**=TIMER** The system waits 6 seconds before the next screen is displayed.

**NEWSCREEN** This operand controls page turning.

**=YES** After statement input, the screen is cleared before the function output occurs.

**=NO** The screen is not cleared after statement input.

**?** The current value is logged.

## TOC Output library directory

TOC outputs the directory entries of the specified members or of the entire library. With the aid of processing operand REFERENCE the directory can be limited to the members containing a certain reference name.

Operation	Operands
TOC[x]	[ { member [ (lib) ] } , ... ] [ (lib) ]

TOCx	Statement name with specification of member type. All member types are permitted: S, M, R, P, J, C, D, X, H, L, F, U  * Stands for all member types (not permitted for tape libraries)  The member type need not be specified if the appropriate member type has been defined in the TYPE processing operand.
member	Member designation or multiple selection.
lib	Short designation of the input library.

### Processing operands

TYPE	Defines the member type, if no type is specified in the statement itself.
REFERENCE	Defines which reference names the members to be output must have. If the reference condition is not satisfied, the member is not output. REFERENCE is permitted for member type R only. After a REFERENCE definition, R must be specified with TOC.
SORT	Defines the sort criteria for output of the directory.  Default value: The directory is sorted by name, version number and date.
TOC	Controls the format of the directory log for program libraries.  Default value: The format is redefined for each member type in accordance with the length of the member names.



**LINE** Defines the number of lines and columns on a log page for output to system file SYSLST or to a member.

Directories that cannot be sorted because of their size can be sorted contiguously by setting job switch 9.

*Example*

```
/SET-FILE-LINK FILE-NAME=A.LIB, LINK-NAME=LIB001
/START-PROGRAM $LMS
$LIB INOUT.LIB, BOTH
$TOC* *
$PAR SORT=V
$TOCS (1)
$END
```

The /SET-FILE-LINK command is used to assign source library A.LIB. Program library INOUT.LIB is assigned as the standard I/O library. Since this is a program library, \* may be entered as the member type in TOC. With all other libraries, \* cannot be entered as a member type for TOC. By means of the second TOC statement, all S-type members can be output from A.LIB, sorted by version.

*Note*

TOC or TOC\* \*/\* must be specified in order to obtain the complete directory of a program library (all members with all versions). This function runs on all library types on which type "\*" is permitted.

## UPD Correct object and load modules and LLMs

UPD corrects the specified member of the assigned input library. The corrected member is then written to the assigned output library, possibly with a new member designation.

UPD includes various substatements for corrections to object and load modules and LLMs. The substatements are read directly after UPD until \*END is encountered in the statement stream.

*No* continuation lines are possible in substatements.

Input and output libraries must not be identical where sequential libraries are concerned. For other types of libraries input and output libraries may be identical.

Three UPD formats are available:

### Format 1: Correct object modules

Operation	Operands
UPD[R]	member [ (lib) ] [>memberu]

### Format 2: Correct load modules (BS2000 phases)

Operation	Operands
UPD[C]	elem [ (lib) ] [>elemu]

### Format 3: Correct LLMs

Operation	Operands
UPD[L]	member [ (lib) ] [>memberu]

**Format 1:** Correct object modules

Operation	Operands
UPD[R]	member [ (lib) ] [>memberu]

UPDR	Statement name with member type R. R may be omitted if member type R has been defined in the TYPE processing operand.
member	Complete designation of the member to be corrected, or multiple selection (no list specification).
lib	Short designation of the input library.
memberu	Designation of the output member, or construction specification.

**Processing operands**

TYPE	Defines the member type if no type is specified in the statement itself.
OVERWRITE	Controls overwriting of identically named members in the output library. If,  input library = output library and member = memberu,  this processing operand will have no effect and the input member is then overwritten.
DESTROY	Defines whether a code for physical deletion is entered in the output member (only possible in conjunction with program libraries).
STRIP	Determines which record types are excluded from correction.

LMS first collects the UPD substatements and checks for

- correct substatement syntax
- uniqueness of corrections (overlapping)
- uniqueness of symbols in the case of renaming.

After \*END, the substatements which have been entered are first checked to see whether they can be executed before an attempt is made to execute them.

## Correction statements for UPDR

## Overview

Correction statement	Function
*BAS baseaddr	Define a base address
*CON controlnumber	Define the cross control number
*COR [csectname,][baseaddr+]address,  $\left[ \left[ \begin{array}{c} \underline{C} \\ \underline{X} \\ \underline{B} \end{array} \right] \right] \text{'searchstring'} = [ := ] \left[ \begin{array}{c} \underline{C} \\ \underline{X} \\ \underline{B} \end{array} \right] \text{'replacementstring'}$ [, ID='ident'] [, CONTROL=number]	Correct text records
*DEL $\left\{ \begin{array}{l} \text{rectype} \\ (\text{rectype}, \dots) \\ \text{TXTP}[, \text{ID}='ident'] \end{array} \right\}$	Delete parts of object modules
*END	Terminate correction input
*ID 'ident'	Define the identification
*INS INCLUDE (module, ...) [, library]	Insert an INCLUDE record
*INV $\left\{ \begin{array}{l} \text{REP} \\ \text{COR}[, \text{ID}='ident'] \end{array} \right\}$	Convert corrections
*NAM $\left\{ \begin{array}{l} \text{CSECT:} \\ \text{ENTRY:} \\ \text{EXTRN:} \\ \text{COMMON:} \end{array} \right\} \text{nameold, namenew}$	Rename symbols
*REM [ID='ident']	Cancel corrections

Correction statement	Function
<pre>*REP [csectname,][baseaddr+]address,        [ [ { C           X           B } ] 'searchstring'=[:=] [ { C           X           B } ] 'replacementstring'        [,CONTROL=number]</pre>	Insert a REP record
<pre>*SET { csectname } [ , PRIV={ Y           N } ] [ , PUBLIC={ Y           N } ]        [ , VISIBLE={ Y           N } ] [ , READONLY={ Y           N } ] [ , PAGE={ Y           N } ]        [ , RESIDENT={ Y           N } ] [ , RMODE={ 24           ANY } ] [ , AMODE={ 24           31           ANY } ]</pre>	Change control section attributes

**Description of the correction statements for object modules**

\*BAS Define base address

\*BAS defines a base address. This address is then added to the address in a subsequent \*COR to form the absolute address in the object module, provided that no base address is explicitly specified in \*COR. The default value is 0.

Operation	Operands
*BAS	baseaddr

baseaddr Defines the base address (in hexadecimal form).  
 $0 \leq \text{baseaddr} \leq 7\text{FFFFFFF}$

\*CON Define cross control number

\*CON defines the cross control number for the entire correction run. If \*CON is specified more than once, the last specification is always the one which is valid.

Control numbers provide more reliability when making and relaying corrections. During the correction run, LMS uses the character string in each correction statement to compute a control number, and from the sum of these numbers LMS in turn computes the cross control number. These values are also determined in test mode. The control numbers are output when the relevant correction statement is logged; the cross control number is output at the end of the correction log. When a correction is relayed or is definitively performed in run mode, the control numbers and the cross control number should also be specified. If corrections are to be carried out, LMS compares the specified numbers with the newly computed ones. If they are found to be not matching, no correction will be performed.

Operation	Operands
*CON	controlnumber

controlnumber Defines the cross control number (in hexadecimal form).  $0 \leq \text{controlnumber} \leq 7\text{FFFFFFF}$

\*COR Correct text records

\*COR corrects text records of an object module and generates a correction journal record (TXTP record) containing the original contents of the text area.

Operation	Operands
*COR	$[csectname, ] [baseaddr+] address,$ $[[\begin{matrix} C \\ X \\ B \end{matrix}]] 'searchstring' = [ := ] [[\begin{matrix} C \\ X \\ B \end{matrix}]] 'replacementstring'$ $[, ID='ident'] [, CONTROL=number]$

- csectname** Specifies the name of a control section (CSECT). If "csectname" is specified, corrections are made within that CSECT only. With new-format advanced prelinked modules (with complete ESD), corrections must always be made via "csectname".
- baseaddr** Defines the base address (in hexadecimal form). This base address only applies for this \*COR. If "baseaddr" is not specified, the address given in \*BAS is assumed.  
 $0 \leq baseaddr \leq 7FFFFFFF$
- address** Defines the relative address.  
"baseaddr" + "address" produce the absolute address in the object module, or the CSECT relative address if "csectname" is specified.  
 $0 \leq baseaddr+address \leq 7FFFFFFF$
- C'searchstring'** Specifies the search string in characters. An apostrophe in the text must be specified in duplicate. "searchstring" may be up to 50 characters in length.
- X'searchstring'** Specifies the search string in hexadecimal form.  
"searchstring" may be up to 50 bytes in length.
- B'searchstring'** Specifies the search string in binary form.  
"searchstring" may be up to 50 characters in length.  
The original text to be compared with the search string is formed from the TXT records existing for this area. If there is more than one text for the same address, the last text is the valid one.
- :=** Search string and replacement string must be of equal length.

- = Search string and replacement string may have different lengths.
- C'replacementstring'  
Specifies the replacement string in characters. An apostrophe in the text must be specified in duplicate. "replacementstring" may be up to 50 characters in length.
- X'replacementstring'  
Specifies the replacement string in hexadecimal form.  
"replacementstring" may be up to 50 bytes in length.
- B'replacementstring'  
Specifies the replacement string in binary form.  
"replacementstring" may be up to 50 characters in length.
- ID='ident'  
Specifies an identifier in character form.  
"ident" may be up to 8 characters in length. This identification applies to this \*COR only. If this operand is omitted, the specification given in \*ID is assumed.
- CONTROL=number  
Defines a local control number (in hexadecimal form).  
 $0 \leq \text{number} \leq \text{FFFF}$

If no TXT record exists or if one exists only for part of the replacement area, LMS will generate a TXT record and include it in the module.

More than one text record may exist for each address (e.g. as a result of ORG statements). In such cases several text records may have to be updated.

No \*REP may be specified within any of the correction statements, since the text corrections would then be overwritten in the course of the subsequent linkage process.



\*DEL Delete parts of object modules

\*DEL excludes the following record types from the input member:

- ISD records
- LSD records
- REP records
- INCLUDE records
- TXTP records
- DSDD records

Operation	Operands
*DEL	<pre> {   rectype   {rectype, ...}   [TXTP[, ID='ident']] </pre>

rectype Defines the record type which is not to be transferred from the input member to the output member. Permissible record types are:

- ISD
- LSD
- REP
- INCLUDE
- TXTP
- DSDD

ID='ident' Specifies an identifier in character form. This identification applies only for this \*DEL.

"ident" may be up to 8 characters in length.

If this operand is omitted, the specification in \*ID applies.

*Note*

REP, INCLUDE and TXTP records should only be deleted after careful consideration.

## UPDR correction statement \*END/\*ID

---

\*END Terminate correction input

\*END concludes the string of correction statements. Afterwards LMS checks all statements to see whether they can be executed before it tries to execute them.

Operation	Operands
*END	

\*ID Define identification

\*ID defines a global identification. It is valid for all statements in which no local identification is specified.

Operation	Operands
*ID	[ 'ident' ]

'ident' Specifies the global identification in characters.  
"ident" may be up to 8 characters in length.  
If this operand is omitted, 8 blanks are assumed by default.

**\*INS** Insert INCLUDE record

\*INS inserts an INCLUDE record into an object module. The INCLUDE record is interpreted by the dynamic binder loader (DBL).

Operation	Operands
*INS	INCLUDE (module,...)[,library]

**module** Name of the object module to be linked in. Up to 10 object modules may be specified. If only one object module is specified, the parentheses may be omitted. "module" may be up to 8 characters in length.

**library** Name of the program or object module library which contains the specified object modules. If this operand is omitted, DBL assumes TASKLIB.

*Notes*

- The specification "(module,...)[,library]" may be up to 71 characters in length.
- LMS neither checks for the existence of the specified object modules nor does it check the library entry.

\*INV Convert corrections

\*INV converts either REP records to text corrections or text corrections to REP records.

**Format 1:** Convert REP records to text corrections

All REP records of the object module are converted to text corrections. As a result processing of REP records is dispensed with when linking and loading take place. Converted REP records are removed from the object module. LMS creates correction journal records for converted text records. For new-format advanced prelinked modules this statement is rejected with an error message.

Operation	Operands
*INV	REP

**Format 2:** Convert text corrections to REP records

Either all text corrections or text corrections under a specific identification in an object module and for which a correction journal record exists are converted to REP records (see \*REP for possible problems with advanced prelinked modules). Thereafter, the correction journal records are deleted. For new-format advanced prelinked modules this statement is rejected with an error message.

Operation	Operands
*INV	COR[, ID='ident']

ID='ident'

Specifies an identification.  
"ident" may be up to 8 characters in length.

If this operand is omitted, the specification in \*ID is assumed. If no specification was made in \*ID either, all text corrections are converted.

**\*NAM** Rename symbols

\*NAM changes the name of a CSECT, ENTRY, EXTRN or COMMON. Each renaming results in a modification of the ESD records. LMS checks for the uniqueness of names within all ESD records, rejecting a new name if that name already exists (in contrast to LMR).

Operation	Operands													
*NAM	<table style="border: none;"> <tr> <td style="border: none;">{</td> <td style="border: none;">CSECT:</td> <td style="border: none;">}</td> <td rowspan="4" style="border: none;">}nameold, namenew</td> </tr> <tr> <td style="border: none;">{</td> <td style="border: none;">ENTRY:</td> <td style="border: none;">}</td> </tr> <tr> <td style="border: none;">{</td> <td style="border: none;">EXTRN:</td> <td style="border: none;">}</td> </tr> <tr> <td style="border: none;">{</td> <td style="border: none;">COMMON:</td> <td style="border: none;">}</td> </tr> </table>	{	CSECT:	}	}nameold, namenew	{	ENTRY:	}	{	EXTRN:	}	{	COMMON:	}
{	CSECT:	}	}nameold, namenew											
{	ENTRY:	}												
{	EXTRN:	}												
{	COMMON:	}												

nameold            Old name of the symbol. The name must be specified in full.

namenew           New name of the symbol. The name must be specified in full.  
 "namenew" may be up to 8 characters in length.

*Note*

Masked CSECT names may also be changed.

**\*REM** Cancel corrections

\*REM cancels all text corrections or text corrections of a specific ID for which a correction journal record exists. The correction journal records are then closed.

Operation	Operands
*REM	[ID='ident']

ID='ident'        Specifies the local ID in characters.  
 "ident" may be up to 8 characters in length.

If this operand is not specified, the specification for \*ID is valid.

## UPDR correction statement \*REP

---

\*REP Insert REP record

\*REP adds REP records to the object module. These REP records are interpreted by the dynamic binder loader (DBL).

Operation	Operands
*REP	$[csectname, ] [baseaddr+] address,$ $[[\begin{matrix} C \\ X \\ B \end{matrix}]] 'searchstring' [=:] [[\begin{matrix} C \\ X \\ B \end{matrix}]] 'replacementstring'$ $[, CONTROL=number]$

csectname	<p>Specifies the name of a control section (CSECT).            For new-format advanced prelinked modules this specification is not permitted. These can only be corrected via absolute addresses.</p>
baseaddr	<p>Defines the base address (in hexadecimal form). This base address applies for this *REP only. If "baseaddr" is omitted, the value specified in *BAS applies.</p> <p><math>0 \leq baseaddr \leq FFFFF</math></p> <p>If a base address greater than FFFFF is specified, the specification is rejected and an error message is issued.</p>
address	<p>Defines the relative address.            "baseaddr" + "address" produces the absolute address in the object module.</p> <p><math>0 \leq baseaddr + address \leq FFFFF</math></p> <p>The address must lie within the module text area.</p>
<u>C'searchstring'</u>	<p>Specifies the search string in character form. An apostrophe in the text must be specified in duplicate.            "searchstring" may be up to 50 characters in length.</p>
X'searchstring'	<p>Specifies the search string in hexadecimal form.            "searchstring" may be up to 50 bytes in length. For new-format advanced prelinked modules, "searchstring" is ignored, i.e. no check is made for old contents.</p>

- B'searchstring' Specifies the search string in binary form.  
 "searchstring" may be up to 50 characters in length. For new-format advanced prelinked modules, "searchstring" is ignored, i.e. no check is made for old contents.
- The original text for comparison with the search string is formed from the TXT records existing for this area. When there is more than one text for the same address, the last text is valid.
- := Search string and replacement string must be of equal length.
- = Search string and replacement string may have different lengths.
- C'replacementstring' Specifies the replacement string in characters. An apostrophe in the text must be specified in duplicate.  
 "replacementstring" may be up to 50 characters in length.
- X'replacementstring' Specifies the replacement string in hexadecimal form.  
 "replacementstring" may be up to 50 bytes in length.
- B'replacementstring' Specifies the replacement string in binary form.  
 "replacementstring" may be up to 50 characters in length.
- CONTROL=number Defines the local control number (in hexadecimal form).  
 $0 \leq \text{number} \leq \text{FFFF}$
- The REP record is inserted only if the control number determined by LMS matches the control number specified here.

*Note*

In contrast to \*COR, LMS does not check whether REP records already exist for the replacement area. The replacement of an advanced prelinked module should always be carried out without CSECT specification and search string, i.e., via relative addresses within the advanced prelinked module. The full range of functions can only be utilized with object modules that are the result of a compilation.

## UPDR correction statement \*SET

---

\*SET Modify control section attributes

\*SET modifies control section attributes.

Operation	Operands
*SET	$\left\{ \begin{array}{l} \text{csectname} \\ * \end{array} \right\} [ , \text{PRIV} = \left\{ \begin{array}{l} \text{Y} \\ \text{N} \end{array} \right\} ] [ , \text{PUBLIC} = \left\{ \begin{array}{l} \text{Y} \\ \text{N} \end{array} \right\} ]$ $[ , \text{VISIBLE} = \left\{ \begin{array}{l} \text{Y} \\ \text{N} \end{array} \right\} ] [ , \text{READONLY} = \left\{ \begin{array}{l} \text{Y} \\ \text{N} \end{array} \right\} ] [ , \text{PAGE} = \left\{ \begin{array}{l} \text{Y} \\ \text{N} \end{array} \right\} ]$ $[ , \text{RESIDENT} = \left\{ \begin{array}{l} \text{Y} \\ \text{N} \end{array} \right\} ] [ , \text{RMODE} = \left\{ \begin{array}{l} 24 \\ \text{ANY} \end{array} \right\} ] [ , \text{AMODE} = \left\{ \begin{array}{l} 24 \\ 31 \\ \text{ANY} \end{array} \right\} ]$

csectname	Name of a control section whose attributes are to be modified.
*	Indicates that the attributes in all control sections are to be modified.
PRIV	
=Y	Specifies that only privileged system routines are allowed to access the specified control sections.
=N	There are no access restrictions.
PUBLIC	
=Y	The specified control sections are shareable.
=N	The specified control sections are not shareable.
VISIBLE	
=Y	The specified control sections are not masked (see the "Binder-Loader-Starter" manual [2]). A secondary name record is created for these sections, and the names are entered in the directory of secondary names.
=N	The specified control sections are masked. No secondary name record is created for these sections, and the names are not entered in the directory of secondary names. Any secondary name record which happens to exist is deleted.
	If all control sections of an object module are masked, a library member without a secondary name entry is created. This object module can be located via the primary name only. LMS issues a warning message to this effect.



The module name can, however, be derived from the initial control section name with the aid of all ESD records, since masked control sections are also used in this case.

*Note*

The linkage editor cannot process object modules which only have masked control sections, e.g. when an object module is excluded with the autolink function.

READONLY

=Y

Indicates that only read access to the specified control sections is permitted while the program is executing.

=N

Enables write access to the specified control sections even while the program is executing.

PAGE

=Y

Indicates that the specified control sections are to be aligned on the page boundary, i.e. the load address should be a multiple of decimal 4096 or hexadecimal 1000.

=N

Does not take page boundaries into account. The control sections always start at the next doubleword address produced during the linkage process.

RESIDENT

=Y

Indicates that the specified control sections are to be loaded in class 3 memory and stored there.

=N

Indicates that the specified control sections are not to be loaded in class 3 memory.

RMODE

=24

Indicates that the specified control sections are to be loaded to the address area below the 16 MB limit.

=ANY

No limitation exists.

AMODE

=24

Indicates that the specified control sections are to be executable in 24-bit mode.

=31

Indicates that the specified control sections are to be executable in 31-bit mode.

=ANY

The mode is freely selectable.

*Note*

Specification of at least one attribute is mandatory, otherwise an error message is issued.

**Old UPDR format**

The old UPDR format is supported for reasons of compatibility only.

Operation	Operands
UPD[R]	member[ (lib) ] [ ; number ]

See the new UPDR format for an explanation of UPDR, "member" and "lib".

number                    Specifies the cross control number in hexadecimal form.

$$0 \leq \text{number} \leq \text{FFFFFF}$$

See the new UPDR format for the effect of the processing operands.

**Correction journal**

For each existing record that is to be updated, LMS writes a correction journal record (TXTP record). This correction journal record contains the original text of the corrected record.

**Description of the correction statement using the old format**

```
1 (address) [bitnumber] [, [[v] 'text1' = [: =]] [v] 'text2' ] [, number]
```

or the short form:

```
1 (address) bitnumber [, number]
```

If only one bit is to be set to 1, the short form may be used. Correction statements comprising simply an address specification are not permitted.

Address type:            Module address.  
address                    Hexadecimal address, up to 8 digits.  
                              This can be taken from the language processor listing or from the  
                              module listing without conversion. Leading zeros may be omitted.

bitnumber                Decimal number, up to 3 digits.  
                              This specifies the number of a bit in the field defined by "address".  
                              The bits are numbered from the left, beginning with 1.

v	X	Text is specified in hexadecimal form
	B	Text is specified in binary form
	v not specified	Text is alphanumeric or consists of one special character.
text1		Search string, up to 50 characters. An apostrophe in an alphanumeric text must be specified in duplicate. If the text is hexadecimal, an even number of characters must be specified. The search string is always compared with the original text in the object module.
=:=		Search string and replacement string must be of equal length.
=		Search string and replacement string may be of different lengths.
text2		Replacement string, as "text1"
number		Control number, hexadecimal, 4-digit.

*Note*

If a new-format advanced prelinked module is to be corrected with this correction statement, the statement will be rejected with an error message.

If a bit number is specified, v=B must be set and vice versa.

The search (replacement) string area is formed from the specified address and the length of the particular text specified in the correction statement. If a search string is specified, it is compared with the search string area in the member; when a match is found, the correction is made and, depending on the value of processing operand STRIP, a correction journal record is created. If an address in the member occurs more than once, the last text is checked; when a match is found, all the text positions of the member that occur in the replacement string area will be changed.

**Format 2:** Correct load modules (BS2000 phases)

Operation	Operands
UPD[C]	member[ (lib) ] [>memberu]

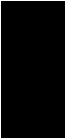
- UPDC            Name of statement with member type C.  
C may be omitted if member type C has been defined in the TYPE processing operand.
- member        Full designation of the member to be corrected. Multiple selection is not permitted (no list specification).
- lib            Short designation of the input library.
- memberu       Designation of the output member, or construction specification.

**Processing operands**

- TYPE           Defines the member type if no type is specified in the statement itself.
- OVERWRITE    Controls overwriting of identically named members in the output library. If,
 

```
input library = output library and
member = memberu,
```

 this processing operand will have no effect and the input member is overwritten.
- DESTROY       Defines whether a code for physical deletion is entered in the output member (only possible in conjunction with program libraries).
- STRIP         Defines which record types are excluded from the correction process.



### Correction statements for UPDC

#### Overview

Correction statement	Function
*BAS baseaddr	Define a base address
*CON controlnumber	Define the cross control number
*COR [segment,][baseaddr+]address, $\left[ \left[ \begin{array}{c} \underline{C} \\ X \end{array} \right] \right] \text{'searchstring'} = [ := ] \left[ \begin{array}{c} \underline{C} \\ X \end{array} \right] \text{'replacementstring'}$ [, ID='ident'] [, CONTROL=number]	Correct text records
*DEL TXTP[, ID='ident']	Delete correction journal records
*END	Terminate correction input
*ID 'ident'	Define the identification
*REM [ID='ident']	Cancel corrections
*SEG $\left\{ \begin{array}{l} \text{segment} \\ \%ROOT \end{array} \right\}$	Define a segment

**Description of the correction statements for load modules**

\*BAS Define base address

\*BAS defines a base address. The base address is then added to the address in a subsequent \*COR to form the absolute address in the load module, provided that no base address is explicitly specified in \*COR. The default value is 0.

Operation	Operands
*BAS	baseaddr

baseaddr Defines the base address (in hexadecimal form).

$$0 \leq \text{baseaddr} \leq 7\text{FFFFFFF}$$

\*CON Define cross control number

\*CON defines the cross control number for the complete correction run. If \*CON is specified more than once, the last specification applies.

Operation	Operands
*CON	controlnumber

controlnumber Defines the cross control number (in hexadecimal form).

$$0 \leq \text{controlnumber} \leq 7\text{FFFFFFF}$$

## UPDC correction statement \*COR

---

\*COR Correct text records

\*COR corrects text records within a segment and produces a correction journal record which contains the original contents of the text area.

Operation	Operands
*COR	$[segment] [baseaddr+] address,$ $[[\left\{ \begin{array}{c} C \\ X \end{array} \right\}]] 'searchstring' = [ := ] [[\left\{ \begin{array}{c} C \\ X \end{array} \right\}]] 'replacementstring'$ $[, ID='ident'] [, CONTROL=number]$

segment	<p>Specifies the name of the segment to be corrected. Corrections can only take place within a segment, and not beyond the boundaries of a segment. "segment" may be up to 8 characters in length.</p> <p>The segment name specified here has priority over the name defined in *SEG. %ROOT is used to select the root segment. If this operand is omitted, the specification in *SEG is assumed.</p>
baseaddr	<p>Defines the base address (in hexadecimal form). This base address applies for this *COR only. If "baseaddr" is omitted, the value specified in *BAS applies.</p> <p><math>0 \leq baseaddr \leq 7FFFFFFF</math></p>
address	<p>Defines the relative address. "baseaddr" + "address" produces the absolute address in the load module.</p> <p><math>0 \leq baseaddr + address \leq 7FFFFFFF</math></p>
<u>C'searchstring'</u>	<p>Specifies the search string in characters. An apostrophe in the text must be specified in duplicate. "searchstring" may be up to 50 characters in length.</p>
X'searchstring'	<p>Specifies the search string in hexadecimal form. "searchstring" may be up to 50 bytes in length.</p>
:=	Search string and replacement string must be of equal length.
=	Search string and replacement string may have different lengths.



C'replacementstring'

Specifies the replacement string in characters. An apostrophe in the text must be specified in duplicate.  
"replacementstring" may be up to 50 characters in length.

X'replacementstring'

Specifies the replacement string in hexadecimal form.  
"replacementstring" may be up to 50 bytes in length.

ID='ident'

Specifies an identifier in characters.  
"ident" may be up to 8 characters in length.

This identification is valid for this \*COR only. If this operand is omitted, the specification in \*ID is assumed.

CONTROL=number

Defines a local control number (in hexadecimal form).

$0 \leq \text{number} \leq \text{FFFF}$



## UPDC correction statement \*DEL/\*END\*ID

---

**\*DEL** Delete correction journal records

\*DEL excludes correction journal records (TXTPs) from the input member.

Operation	Operands
*DEL	TXTP[,ID='ident']

**ID='ident'** Specifies an identifier in characters.  
"ident" may be up to 8 characters in length.  
This identification is valid for this \*DEL only. If this operand is not specified, the specification in \*ID is assumed.

**\*END** Terminate correction input

\*END concludes the string of correction statements. Afterwards LMS checks all statements to see whether they can be executed before it attempts to execute them.

Operation	Operands
*END	

**\*ID** Define identification

\*ID defines a global identification. It is valid for all statements for which no local identification has been specified.

Operation	Operands
*ID	['ident']

**'ident'** Specifies the global identification in characters.  
"ident" may be up to 8 characters in length.  
If the operand is omitted, 8 blanks are assumed as the default value.

\*REM Cancel corrections

\*REM cancels either all text corrections or text corrections under a specific identification for which a correction journal record exists. The correction journal records are then deleted.

Operation	Operands
*REM	[ ID='ident' ]

**ID='ident'** Specifies the local identification in characters. "ident" may be up to 8 characters in length.  
 If this operand is omitted, the specification in \*ID is assumed.

\*SEG Define segment

\*SEG defines a segment of a load module which is to be corrected with the aid of a subsequent \*COR.

Operation	Operands
*SEG	{ segment } { %ROOT }

**segment** Specifies the name of the segment to be corrected. "segment" may be up to 8 characters in length.  
**%ROOT** Specifies that the root segment is to be corrected.

**Format 3:** Correcre LLMs

Operation	Operands
UPDL	member[(lib)][>memberu]

- UPDL            Name of statement with member type L.
- member        Complete member designation of the member to be corrected or a multiple selection.
- lib             Short designation of the input library.
- memberu       Member designation of the output member or construction specification.

**Processing operands**

- CSECT         Defines the base for displacement in the \*COR substatement.
- OVERWRITE    Specifies overwriting of output library members with the same name. This operand is however, not interpreted if
 

```
input library = output library and
                member = memberu
```

 The input member is then overwritten.
- PATH          Defines the base for displacement in the \*COR substatement.
- SLICE         Defines the base for displacement in the \*COR substatement.
- STRIP         Defines the type of records to be excluded from correction.

**Correction statements for UPDL**

## Overview

Correction statement	Function
<pre>*COR [csect,]displ,       {C}       [ [ {X} ] 'searchstring' = [ := ] [ {X} ] 'replacementstring'         {B}       [, ID='ident' ] [, CONTROL=number]</pre>	Correct text records
<pre>*DEL TXTP, [ID='ident']</pre>	Delete correction journal records
<pre>*END</pre>	Terminate input of corrections
<pre>*ID ['ident']</pre>	Define the ID
<pre>*REM [ID='ident']</pre>	Cancel corrections

**Description of the correction statements for LLMs**

\*COR Correct text records

\*COR corrects text records of an LLM.

Operation	Operands
*COR	<pre>[csect, )displ,     [ [ {X} ] 'searchstring' [=[:=]] [ {X} ] 'replacementstring'     [ , ID='ident' ] [ , CONTROL=number ]</pre>

- csect** CSECT name with a length of 32 characters. If CSECT is specified, all the CSECTs with the specified name are corrected.
- When evaluating the name, the priority is as follows:
1. "csect" has been specified in \*COR; the CSECT processing operand is ignored.
  2. PAR CSECT=xxx; only the processing operand PATH or SLICE may be set.
  3. PAR PATH=xxx: the SLICE processing operand must not be set.
  4. PAR SLICE=xxx: the PATH processing operand must not be set.
- displ** Defines the relative address.  
 "displ" produces the absolute address in the LLM or the CSECT relative address if "csect" is specified.  
 $0 \leq \text{displ} \leq 7\text{FFFFFFF}$
- C'searchstring'** Specifies the search string in characters. An apostrophe in the text must be specified in duplicate. "searchstring" may be up to 50 characters in length.
- X'searchstring'** Specifies the search string in hexadecimal form. "searchstring" may be up to 50 bytes in length.

- B'searchstring' Specifies the search string in binary form.  
 "searchstring" may be up to 50 characters in length.  
 The original text to be compared with the search string is formed from the TXT records existing for this area. If there is more than one text for the same address, the last text is the valid one.
- := Search string and replacement string must be of equal length.
- = Search string and replacement string may have different lengths.
- C'replacementstring' Specifies the replacement string in characters. An apostrophe in the text must be specified in duplicate. "replacementstring" may be up to 50 characters in length.
- X'replacementstring' Specifies the replacement string in hexadecimal form.  
 "replacementstring" may be up to 50 bytes in length.
- B'replacementstring' Specifies the replacement string in binary form.  
 "replacementstring" may be up to 50 characters in length.
- ID='ident' Specifies an identifier.  
 "ident" may be up to 8 characters in length.  
 This identification applies to this \*COR only. If this operand is omitted, the specification given in \*ID is assumed.
- CONTROL=number Defines a local control number (in hexadecimal form).  
 LMS calculates the control number for each \*COR statement.  
 $0 \leq \text{number} \leq \text{FFFF}$

## UPDL correction statement \*DEL/\*END

---

\*DEL Delete correction journal records

Operation	Operands
*DEL	TXTP[, ID='ident']

**TXTP** Correction journal records are to be excluded from the input member.

**ID='ident'** Specifies an identifier.  
"ident" may be up to 8 characters in length.  
This identification is valid only for this \*DEL. If this operand is omitted, the specification in \*ID applies.

\*END Terminate correction input

\*END concludes the string of correction statements. Afterwards LMS checks all statements to see whether they can be executed before it tries to execute them.

Operation	Operands
*END	



**\*ID** Define identification

\*ID defines a global identification. It is valid for all statements in which no local identification is specified.

Operation	Operands
*ID	[ 'ident' ]

'ident' Specifies the global identification.  
 "ident" may be up to 8 characters in length.  
 If this operand is omitted, 8 blanks are assumed by default.

**\*REM** Cancel corrections

\*REM cancels either all text corrections or text corrections under a specific identification for which a correction journal record exists. The correction journal records are then deleted.

Operation	Operands
*REM	[ ID='ident' ]

ID='ident' Specifies the local identification.  
 "ident" may be up to 8 characters in length.  
 If this operand is omitted, the specification in \*ID applies.

## USE Branch to user programs

USE permits LMS to branch to a user routine prior to processing a member record.

The function is permitted:

- when listing members of the type R, S, M, P, J, D and X using LST, and
- when comparing members of the type S, M, P, J, D and X using COM.

Before LMS processes the member record, the following actions are possible:

- update the current member record
- insert records via the user routine
- exclude the current member record from processing

The user routine is informed of start and end of member so as to enable the user to insert records before the first and after the last member record.

For COM, two user exits are provided: one for the primary member and one for the secondary member. The user exits for LST and COM can be defined simultaneously.

If several USE statements with the same user exit are defined, the last one specified applies.

Furthermore, it is possible to define the libraries LMS is to use for the connection of the EDT, EDOR and FMS subroutines.

Operation	Operands
USE	$\left[ \begin{array}{l} \left\{ \begin{array}{l} \text{LST} \\ \text{COMP} \\ \text{COMS} \end{array} \right\} = \left[ \begin{array}{l} \left\{ \begin{array}{l} \text{entry} \\ * \\ \text{libraryyu} \end{array} \right\} (\text{entry}) \end{array} \right] \\ \\ \left\{ \begin{array}{l} \text{EDTLIB} \\ \text{EDORLIB} \\ \text{FMSLIB} \end{array} \right\} = \text{librarys} \\ \\ ? \end{array} \right]$

- USE            Statement name.
- LST            User exit for the LST function.

COMP	User exit for the COM function when the primary member is processed.
COMS	User exit for the COM function when the secondary member is processed.
entry	Name of the entry point for the user routine, up to 8 characters in length. "entry" must not start with the character string "LMS".
*	The user routine is dynamically loaded from the EAM area.
libraryu	Name of the library, up to 54 characters in length, in which the user routine is stored.
EDTLIB	User exit for the EDT subroutine.
EDORLIB	User exit for the EDOR subroutine.
FMSLIB	User exit for the FMS subroutine.
librarys	Name of the library, up to 54 characters in length, in which the appropriate subroutine is to be found.
?	The current value is logged.

#### **Dynamically loading the user program**

The user program is not loaded dynamically from the library or EAM area until it is used for the first time. It is always loaded into user-own class 6 memory. If the specification "\*" or "library" is omitted, the user program will first be sought in a private TASKLIB (assigned with /SET-TASKLIB LIBRARY=library), if present, and then in the system TASKLIB (\$TASKLIB). The same applies if "\*" or "library" is specified but the user program has not been found there.

#### **Deactivating the user exit**

If "entry" and "library" are not specified, LMS will deactivate the user exit. It will no longer be used when the next corresponding LST or COM references it. However, the user routine will not be deactivated, as it may still be required for other purposes. This means that the user program need not be linked again for the next USE having the same entry point.

**User exit interface**

## Register conventions

When the user program is called, LMS will take account of the following register conventions:

Register 1:     Address of a parameter list  
Register 13:    Address of the save area (18 words)  
Register 14:    Return address  
Register 15:    Address of entry point

## Structure of the parameter list

The parameter list consists of 5 words:

DC A (job description)  
DC A (response description)  
DC A (record to be transferred)  
DC A (library name)  
DC A (member designation)

The first two addresses are provided by LMS. This means that the user can only supply the response description to the address that is specified by LMS. The record address can be supplied by both LMS and the user.

## 1st word: Job description

The job description for the user subroutine consists of 3 bytes and can have the following contents:

C'BOE'            Beginning of member (element)  
C'REC'            Record ready to be processed  
C'EOE'            End of member (element)

## 2nd word: Response description

The response description issued by the user subroutine consists of three bytes and may have the following contents:

C'CON'            LMS processes the record whose address is in the parameter list on return from the user program, and then submits the next member record or EOE.  
C'DEL'            LMS bypasses the last submitted member record and branches back to the user subroutine at the next member record or at EOE.

C'INS' LMS first processes the record submitted by the user, and then returns to the user subroutine at the member record submitted previously or at EOE. This is repeated until the response DEL or CON is given. This means that, if LMS submits record i, the records returned with INS are processed before record i.

The following job responses are possible:

Job	Response
BOE	Irrelevant
REC	CON, DEL, INS
EOE	INS, CON

If the response is incorrect, the LMS function is aborted and an error message is issued.

The diagram illustrates the communication between LMS and the user subroutine.

3rd word: Record

This contains the address of the record that is passed by LMS to the user subroutine. LMS does not provide a value for this record address until the REC job is received. Prior to this the record address is not supplied with a value, i.e. it contains X'00000000'.

When the user subroutine returns with the response CON or DEL, the same record address may be used. If the user wants to insert records, he must use a record buffer he has defined himself. The records exchanged between LMS and the user subroutine start with a 4-byte record length field. If processing operand FORMAT=P is set, the 5th byte must be a valid feed control character in order to produce lists with the aid of LST.

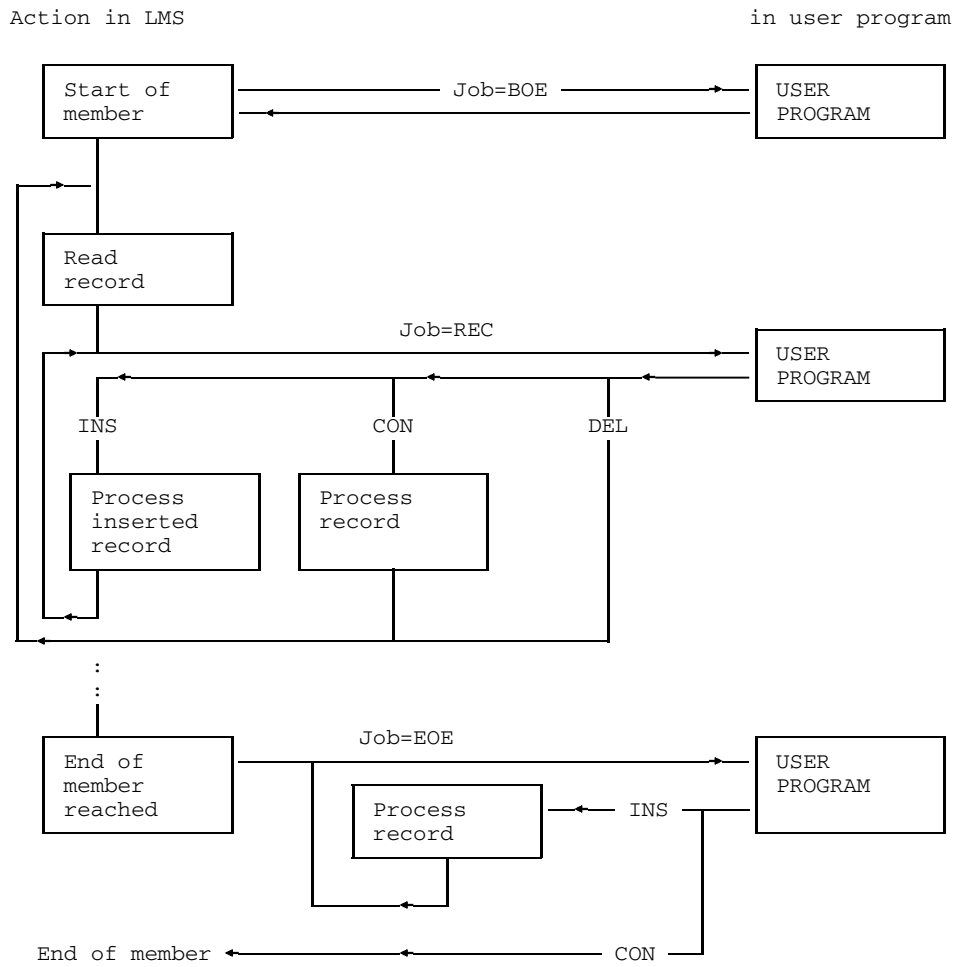
4th word: Library name

This word contains the address of the library name. The library name starts with a record length field of two bytes.

5th word: Member designation

This word contains the address of the member designation. The member designation format is: record length field of two bytes, followed by (type)membername/version[(variantnumber)]/date.

Diagram:



See page 285 for an example of the user exit.

## \$ Output statement buffer

The LMS statements can be entered in block mode. This means that they need not be entered at the display terminal one at a time in order to be processed; instead a common data transfer of several statements can be started (see section on "Entry of blocked statements", page 84).

In block mode, the statements are written to a statement buffer. Each sequence of statements (other than \$) causes the buffer to be updated.

\$ logs the last statement buffer, which may be changed or reentered at the display terminal.

Operation	Operands
\$	

\$ Statement name.

Following a syntax error, or another severe error, processing of the current statement buffer is aborted in interactive mode. LMS remains in run mode and expects new statements from the display terminal.

\$ enables the user to display the entire statement buffer, remove already executed statements, correct the invalid statement and reenter the buffer.





## Processing operands

The LMS run can be controlled by means of processing operands.

If a processing operand is to control a certain function, it must be set prior to the appropriate statement.

Processing operands are set with the aid of PAR:

Operation	Operands
PAR	$\left[ \left\{ \begin{array}{l} \text{parname} = \left[ \left\{ \begin{array}{l} \text{parvalue} \\ ? \end{array} \right\} \right] \\ ? \end{array} \right\} , \{ \dots \} \right]$

See page 205 ff for an overview of the processing operands in table form. The processing operands are described in alphabetical order on page 209 ff.

The names of the processing operands and operand values may be abbreviated, provided that the abbreviation is unique among all the processing operands. For example, BASE can be abbreviated to just B, but STRING may be shortened only to STRIN (to avoid confusion with STRIP).

The following operands affect the overall operation of LMS, but not individual functions:

ERRCONS	Error messages on the display terminal
TERMINATE	Termination procedure in the event of an error
NEWFORM	Page control for logs
LCASE	Lower/upper case
LINE	Number of lines per log page

## Processing operands

---

The following processing operands affect both the entire LMS run and also individual functions:

TEST	Test mode, no library modifications; affects the function RST in addition to the entire LMS run.
LOG	Logging of statements; also affects the logging of the functions COR and NUM.

The other processing operands affect only individual functions. The descriptions of the functions indicate which operands affect which functions. A table listing the effects of processing operands is provided on page 63.

If PAR is specified without any processing operand, then all processing operands will be set to their default values.

If a processing operand without operand value is specified (e.g. PAR=B), the processing operand concerned will be reset to its default value.

## Table of processing operands

The following table provides an overview of all processing operands, in alphabetical order.

Processing operand	Application
B[ASE] = { $\left. \begin{array}{l} \text{baseaddr} \\ \underline{0} \\ ? \end{array} \right\}$	Define a base address
CH[ECK] = { $\left. \begin{array}{l} [\text{start}] [/\text{length}] \\ \underline{NO} \\ ? \end{array} \right\}$	Define the check field in input records
COM[PARE] = { $\left. \begin{array}{l} [[\text{start}]/\text{length}] [/\text{compare}[\text{number}]] [/\text{list}] [/\text{COR}] \\ \underline{1/72/L/MED} \\ ? \end{array} \right\}$	Control the comparison function
CS[ECT] = { $\left. \begin{array}{l} \text{name} \\ ? \end{array} \right\}$ }	Define the CSECT
DES[TROY] = { $\left. \begin{array}{l} \text{YES} \\ \underline{NO} \\ ? \end{array} \right\}$	Control physical deletion
E[RRCONS] = { $\left. \begin{array}{l} \text{NO} \\ 2 \\ 4 \\ \underline{ALL} \\ \text{YES} \\ ? \end{array} \right\}$	Output messages to SYSOUT
FC[BTYPE] = { $\left. \begin{array}{l} \text{ISAM} \\ \text{SAM} \\ \underline{STD} \\ \text{CAT} \\ ? \end{array} \right\}$	Define FCB type of output file
FO[RMAT] = { $\left. \begin{array}{l} \text{C} \\ \text{X} \\ \underline{SYMBOLIC} \\ \text{XC} \\ \text{REC} \\ \text{P} \\ ? \end{array} \right\}$	Define record format

## Processing operands

Processing operand	Application	
$I[NFO] = \left[ \begin{array}{l} \underline{ALL} \\ SUMMARY \\ (rectype, \dots) \\ rectype([[\#recstart] \left\{ \begin{array}{l} \{-\#recend\} \\ \{:\#number\} \end{array} \right\}]) \\ \left. \begin{array}{l} TXT([[\#addrstart] \left\{ \begin{array}{l} \{-\#addrend\} \\ \{:\#length\} \end{array} \right\}]) \\ TXTP([[\#identl'] \{-\#identu'\}]) \\ \left. \begin{array}{l} \left\{ \begin{array}{l} PHY[SICAL] \\ LOGICAL(\left\{ \begin{array}{l} \{NEXT\} \\ \{ALL\} \end{array} \right\}) \end{array} \right\} \end{array} \right\} \\ ? \end{array} \right]$	Define scope of output	
$K[EY] = \left\{ \begin{array}{l} \underline{YES} \\ \underline{NO} \\ ? \end{array} \right\}$	Transfer ISAM key and other file attributes	
$LC[ASE] = \left\{ \begin{array}{l} \underline{YES} \\ \underline{NO} \\ ? \end{array} \right\}$	Convert lowercase to uppercase	
$LIN[E] = \left\{ \begin{array}{l} [lines] [/columns] \\ \underline{60/132} \\ ? \end{array} \right\}$	Define number of lines/columns per log page	
$LO[G] = \left\{ \begin{array}{l} \underline{MAX} \\ \underline{MED} \\ \underline{MIN} \\ ? \end{array} \right\}$	Log statements	
$LS[T] = \left\{ \begin{array}{l} \left\{ \begin{array}{l} \underline{TXT} \\ \underline{REC} \end{array} \right\} [ / \left\{ \begin{array}{l} \underline{NUM} \\ \underline{NO [NUM]} \end{array} \right\} ] \\ \underline{PRT} \\ \underline{NO} \\ ? \end{array} \right\}$	$LS[T] = \left\{ \begin{array}{l} \underline{REC} [ / \left\{ \begin{array}{l} \underline{NUM} \\ \underline{NO [NUM]} \end{array} \right\} ] \\ \underline{ALL} \\ \underline{TXT} \\ \underline{UPD} \\ \underline{SYM} \\ \underline{NO} \\ ? \end{array} \right\}$	Define scope and mode of listing of members

Processing operand	Application
$N[EWFORM] = \left[ \begin{array}{c} \{ YES \\ NO \\ \underline{3} \\ \text{number} \\ ? \} \end{array} \right]$	Control form feed
$O[VERWRITE] = \left[ \begin{array}{c} \{ YES \\ ONLY \\ NO \\ \underline{V} \\ D \\ EXTEND \\ ? \} \end{array} \right]$	Overwrite identically named members
$PA[TH] = \left[ \begin{array}{c} \{ \text{name} \\ ? \} \end{array} \right]$	Define the pathname
$PH[ASE] = \left[ \begin{array}{c} \{ PK \\ NK \\ ? \} \end{array} \right]$	Define the phase format
$RA[NGE] = \left[ \begin{array}{c} \{ [\text{start}] [/\text{length}] \\ \underline{NO} \\ ? \} \end{array} \right]$	Define the check field in output records
$RE[FERENCE] = \left[ \begin{array}{c} \left\{ \begin{array}{c} \text{name} \\ \{ ([\text{name}], [ \begin{array}{c} \{ CSECT \\ ENTRY \\ ALL \} \} ]) \end{array} \} \right\} \\ ? \end{array} \right]$	Define reference conditions
$SE[GMENT] = \left[ \begin{array}{c} \left\{ \begin{array}{c} \text{overlay} \\ \%ROOT \\ \%ALL \\ ? \end{array} \right\} \end{array} \right]$	Define segments of a load module
$SL[ICE] = \left[ \begin{array}{c} \left\{ \begin{array}{c} \text{name} \\ ? \end{array} \right\} \end{array} \right]$	Define the slice
$SO[RT] = \left[ \begin{array}{c} \left\{ \begin{array}{c} U \\ [N] [V] [D] \\ R \\ ? \end{array} \right\} \end{array} \right]$	Sort directory

## Processing operands

Processing operand	Application
$\text{STRIN[G]} = \left[ \begin{array}{l} \text{'characterstring'} \\ \text{NAME} \\ \text{KEY} \\ \text{NO} \\ \text{?} \end{array} \right]$	Define a string in the check field of output records
$\text{STRIP} = \left[ \begin{array}{l} \text{rectype} \\ \text{(rectype, ...)} \\ \text{YES} \\ \text{NO} \\ \text{?} \end{array} \right]$	Suppress records
$\text{SU[M]} = \left[ \begin{array}{l} \text{YES} \\ \text{NO} \\ \text{?} \end{array} \right]$	Generate comparison statistics
$\text{TER[MINATE]} = \left[ \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ \text{?} \end{array} \right]$	Set termination procedure for error conditions
$\text{TES[T]} = \left[ \begin{array}{l} \text{YES} \\ \text{NO} \\ \text{ABORT} \\ \text{?} \end{array} \right]$	Activate, deactivate or quit test mode
$\text{TO[C]} = \left[ \begin{array}{l} \text{F} \\ \text{D} \\ \text{L} \\ \text{T} \\ \text{?} \end{array} \right]$	Control the output format for directories of program libraries
$\text{TY[PE]} = \left[ \begin{array}{l} \text{type} \\ \text{?} \end{array} \right]$	Predefine the member type
$\text{V[ALUE]} = \left[ \begin{array}{l} \text{[start] [/inc]} \\ \text{NO} \\ \text{?} \end{array} \right]$	Control numbering in the check field of output records

## PAR BASE      Define base address

The BASE processing operand defines a base address to which the area addresses defined in the INFO processing operand refer. The base address is added to the specified area address.

Operation	Processing operand
PAR	$B[ASE] = \left[ \begin{array}{l} \text{baseaddr} \\ 0 \\ ? \end{array} \right]$

**baseaddr**              Specifies the base address in hexadecimal form:

$$0 \leq \text{baseaddr} \leq 7FFFFFFF$$

If no area addresses are specified in the INFO processing operand, the BASE processing operand is ignored.

**?**                              The current value is logged.

The BASE processing operand affects LST.

## PAR CHECK      Define check field in input records

The CHECK processing operand defines the position and length of the check field in input records. This range is checked for ascending sequence. When members are output to ISAM files the record IDs can be stored as ISAM keys.

Operation	Processing operand
PAR	$\text{CH[ECK]} = \left\{ \begin{array}{l} [\text{start}] [/\text{length}] \\ \underline{\text{NO}} \\ ? \end{array} \right\}$

start	<p>Start of the check field in the input record.            "start" may have the following values:  <math>1 \leq \text{start} \leq 251</math></p>
length	<p>Length of the check field.            "length" may have the following values:  <math>1 \leq \text{length} \leq 16</math></p> <p>If only "start" is specified, it must be in the following range:  <math>65 \leq \text{start} \leq 80</math></p> <p>This is because "length" is calculated as:  <math>\text{length} = 81 - \text{start}</math>.</p> <p>If only "length" is specified, "start" is calculated as:  <math>\text{start} = 81 - \text{length}</math>.</p> <p>If "start" and "length" are specified together, the following applies:  <math>\text{start} + \text{length} \leq 252</math></p>
<u>NO</u>	No check field is defined in the input areas.
?	The current value is logged.

The CHECK processing operand affects:  
 ADD, COM, COR, EDR, EDT, NUM, SEL

### *Note*

The CHECK processing operand only affects COM in cross comparisons.



## PAR CSECT      Specify a CSECT name

This processing operand is used to specify a CSECT name for UPDL and LSTL.

Operation	Processing operand
PAR	$CS[ECT] = [ \left\{ \begin{array}{l} \text{name} \\ ? \end{array} \right\} ]$

- name**      CSECT name with a length of up to 32 characters  
 Only the CSECT with this name is then listed for LSTL.  
 This name is used as the base for displacement in the \*COR substatement of UPDL.
- ?**          The current value is logged.

## PAR COMPARE      Control compare function

The COMPARE processing operand determines the comparison range, the type of comparison and the log format for COM. Two algorithms are provided for comparisons: on the one hand the Heckel algorithm (standard procedure as of LMS Version 1.3A) and, optionally, the cross comparison. An important feature of the Heckel algorithm is that a comparison is only made after all records have been read (no piecemeal comparison). Moreover, only the compared parts of the records are logged and not the complete records as in a cross comparison.

The comparison results may vary on account of the differing procedures.

Operation	Processing operand
PAR	$\text{COM}[\text{PARE}] = \left[ \left\{ \begin{array}{l} [[\text{start}]/\text{length}] [/\text{comparison}[\text{number}] [/\text{list}] [/\text{COR}] \\ 1/72/L/MED \\ ? \end{array} \right\} \right]$

start	<p>Start of the comparison field. "start" may have the following values:</p> <p>Cross comparison:    <math>1 \leq \text{start} \leq 251</math>  Heckel algorithm:    <math>1 \leq \text{start} \leq 32764</math></p>
length	<p>Length of the comparison field. "length" may have the following values:</p> <p>Cross comparison:    <math>1 \leq \text{length} \leq 251</math>  Heckel algorithm:    <math>1 \leq \text{length} \leq 32764</math></p> <p>The value of "start" plus "length" must not exceed 252 (cross comparison) and 32765 (Heckel algorithm) respectively.</p>
comparison	<p>Type of comparison:</p> <p>L: Logical comparison  The comparison fields are compared one character at a time, space characters being skipped.</p> <p>F: Formal comparison  The comparison fields are first checked for equal length. If they differ in length, the fields are logged as unequal. If they have the same length, the fields are compared in their entirety.</p>

number	<p>Synchronization counter.  "number" may have the following values:  <math>1 \leq \text{number} \leq 9</math></p> <p>"number" specifies the minimum number of matching records that must be found for a synchronization attempt to be considered successful (cf. page 106). If fewer than "number" matching records are found after non-matching records, these are assumed to be non-matching and are logged as such.</p> <p><i>Note</i>  If a value has been specified for "number", LMS will switch to cross comparison. PAR COM= permits a return to the Heckel algorithm (see page 50).</p>
list	<p>Scope of logging.</p> <p><b>MAX</b> Detailed comparison log.  All records are logged.  The comparison statistics are output.</p> <p><b>MED</b> Standard comparison log.  Non-matching records are logged in their entirety. For matching records, only range specifications (record number and any record IDs) are logged. The comparison statistics are logged.</p> <p><b>MIN</b> Minimum comparison log.  Only range specifications (record numbers and any IDs) are logged for matching and non-matching records.  The comparison statistics are output.</p> <p><b>SUM</b> No comparison log.  Only the comparison statistics are output.</p> <p><b>NO</b> Neither logging nor comparison log nor comparison statistics.  NO is only meaningful when processing operand SUM is used.</p>
COR	<p>When text members are compared, correction statements for COR are generated by LMS from the records INS and DEL of the comparison log.</p> <p>The correction statements are output to SYSOPT. If the correction statements are not to be output on punched cards, the following command must be issued:</p> <pre>/ASSIGN-SYSOPT TO-FILE=file</pre>

In this case, "file" can again be added to a library as a procedure member, and also be started as a procedure.

See page 269 for an example of this function.

?           The current value is logged.

The COMPARE processing operand affects COM.

## PAR DESTROY Control physical deletion

The DESTROY processing operand controls whether the storage space released when deleting should also be physically destroyed. This means that the data to be deleted will be overwritten with binary zeros.

Only program library members and scratch files created by EDT or EDOR can be physically deleted.

For this operand to be effective, it must be set as soon as the member is incorporated. If it is set later, only variants generated from this point in time will be physically deleted.

Operation	Processing operand
PAR	DES[TROY] = [ { YES } { NO } { ? } ]

- YES** All data to be deleted is physically destroyed, i.e. overwritten with binary zeros, before the storage space is released.
- This means that
- program library non-delta members are physically destroyed when deleted; delta members, however, not until the the last delta member of a tree, i.e. the whole tree is deleted.
  - a code for physical deletion is entered when members are written to program libraries, i.e if such members are deleted later they will be physically destroyed.
  - any scratch files that may have been created are physically destroyed when returning from EDT/EDOR
- This operand value only affects members of program libraries and scratch files generated by EDT or EDOR. When other libraries are involved, it has the same effect as NO.
- NO** Only the storage space is released for all data to be deleted.
- ?** The current value is logged.

The DESTROY processing operand affects:  
ADD, COR, DEL, DUP, EDR, EDT, NAM, NUM, PRT, UPD

## PAR ERRCONS      Output messages to SYSOUT

The ERRCONS processing operand defines which LMS messages are to be additionally output to system file SYSOUT if the tracer log is not output there (i.e. in the case of PRT (LST) or PRT member(lib)).

Operation	Processing operand
PAR	$E[RRCONS] = \left[ \begin{array}{c} \text{NO} \\ 2 \\ 4 \\ \underline{\text{ALL}} \\ \text{YES} \\ ? \end{array} \right]$

- NO                      Does not additionally output any messages to system file SYSOUT.
- 2                        Outputs error messages.
- 4                        Outputs error messages and NOT FOUND messages.
- ALL                    Outputs error messages, NOT FOUND, EXISTING and NO OLD messages.
- YES                     Has the same effect as ALL and is supported for reasons of compatibility only.
- ?                        The current value is logged.

The ERRCONS processing operand affects the entire LMS run.

## PAR FCBTYPE Define FCB type of output file

The FCBTYPE processing operand defines which FCB type is used to create the file to which a member is output. This processing operand is only interpreted for members relating to text, i.e. not for R-type or C-type members.

File attributes specified in the FILE command have priority over specifications in the FCBTYPE processing operand.

File characteristics are also stored in the member by specifying PAR KEY=YES.

Operation	Processing operand
PAR	$FC[BTYPE] = \left\{ \begin{array}{l} ISAM \\ SAM \\ \underline{STD} \\ CAT \\ ? \end{array} \right\}$

- ISAM** Creates an ISAM file with the following attributes:
- If the FCB type ISAM has been entered in the stored file attributes, the file is created in accordance with these attributes.
  - If the FCB type SAM has been entered in the stored file attributes, an ISAM file with KEY-POSITION=5 and KEY-LENGTH=8 is created.
  - If no file attributes are stored, an ISAM file with KEY-POSITION=5 and KEY-LENGTH=8 is created.
- SAM** Creates a SAM file. If the FCB type ISAM is entered in the stored file attributes, a SAM file with or without KEY is created.
- STD** Creates a SAM or ISAM file in accordance with the file attributes stored.
- If no file attributes are stored, the file is created in accordance with the entry in the catalog.
- If the catalog entry is also missing, an ISAM file with KEY-POSITION=5 and KEY-LENGTH=8 is created.

CAT	Creates a SAM or ISAM file in accordance with the entry in the catalog.  If the catalog entry is missing, the file is created in accordance with the file attributes stored. If no file attributes are stored, an ISAM file with KEY-POSITION=5 and KEY-LENGTH=8 is created.
?	The current value is logged.

The FCBTYPE processing operand affects SEL.



## PAR FORMAT Define record format

The FORMAT processing operand defines the record format for listing members with the aid of LST. The following record formats are possible:

- characters
- hexadecimal
- characters and hexadecimal, in sequence
- characters and hexadecimal, one above the other
- characters, where the first character of the record contents is interpreted as a feed control character.

Operation	Processing operand
PAR	$FO[RMAT] = \left\{ \begin{array}{l} C \\ X \\ \underline{SYMBOLIC} \\ XC \\ REC \\ P \\ ? \end{array} \right\}$

- C** Representation in character form.
- X** Hexadecimal representation, only for members of the type
- R and C
  - X, provided that PAM files have been archived in these members.
- 2\*4 4-byte blocks are output to system file SYSLST and 2\*3 4-byte blocks to system file SYSOUT.
- This operand value has the same effect as operand REC on all other records.
- SYMBOLIC** Displays the records for the different member types in different ways:
- | <i>Member type</i> | <i>Record format</i>  |
|--------------------|---|
| S, M, J, D, X      | In characters.  |
| P                  | In characters, where the first character of each record is interpreted as a feed control character. |

	R,L	The ESD, ISD, RLD, TXT, TXTP, REP and END information available to LMS is output in edited form. Other information such as LSD and DSDD, for example, is output in unedited form, i.e. the record length field and the record number, if any, are also output. Consecutive text information is not split up.
	C	In characters and hexadecimal, sequential.
XC		In characters and hexadecimal, sequentially; only for members of the type <ul style="list-style-type: none"> <li>– R, L and C</li> <li>– X, provided that PAM files have been archived in these members.</li> </ul> This operand value has the same effect as REC on all other members.
REC		In characters and hexadecimal, one above the other. This means that, for each member record, two lines are output, with the character display in the first line and the hexadecimal representation in the second line.
P		Representation in characters, where the first character of the contents of each record is interpreted as a feed control character. <p>This type of output is only advisable in conjunction with edited text members or log members.</p> R-type, L-type and C-type members are represented in accordance with the value of the XC operand.
	?	The current value is logged.

The FORMAT processing operand affects LST.

## PAR INFO Define scope of output

The INFO processing operand defines the extent of the record when members are listed with the aid of LST. The following outputs are possible:

- all records
- the most important member data
- specific record types
- output of a specific range within a record type.

Operation	Processing operand
PAR	<pre> I[INFO]=[   {     ALL     SUMMARY     (rectype,...)     rectype[([#recstart][{-#recend}]][:#number])     TXT[([addrstart][{-addrend}]][:length])     TXTP(['identl'][-'identu'])     {       PHY[SICAL]       LOGICAL[({NEXT})]       {ALL}     }   }   ? </pre>

**ALL**

All records of the member are output.

**SUMMARY**

The most important member data is output, i.e.

- for members relating to text (types S, M, J, P, D, X) the existing user record types, the file attributes added using PAR KEY=YES and the number of records are output in the form of a table.
- for R-type members the length of the object module as well as names, lengths and addresses of the CSECTs are output.
- for C-type members the length of the load module as well as names, lengths and address of the segments are output.
- for L-type members the complete logical structure is output.

---

rectype	<p>The specified record type of an R-type, C-type or L-type member (see the "Binder-Loader-Starter" manual [2]) is output. In the case of a member relating to text (types S, M, J, P, D, X) the record type specification is ignored, i.e. all records of the member are output.</p> <p>Potential record types are:</p> <ul style="list-style-type: none"><li>– for R-type members: ESD, ISD, LSD, TXT, RLD, TXTP, REP, INCLUDE, DSDD, REF, END</li><li>REF is used to output all reference names belonging to the object module.</li><li>– for C-type members: ESD, ISD, LSD, TXT, RLD, TXTP</li><li>– for L-type members: ESVD, ESVR, TXT, LRLD, TXTP</li></ul>
restart	<p>Specifies the first record as of which the specified record type is output. For R-type members only.</p> $1 \leq \text{restart} \leq 2147483647$
recend	<p>Specifies the last record up to which the specified record type is output. For R-type members only.</p> $1 \leq \text{recend} \leq 2147483647$
number	<p>Specifies the number of records of the specified record types which is output. For R-type members only.</p> $1 \leq \text{number} \leq 2147483647, \text{ where } \text{restart} + \text{number} \text{ must be } \leq 2147483647.$
addrstart	<p>Specifies in hexadecimal form the start address of the area as of which the specified TXT is output. For R-type, L-type and C-type members only.</p> $0 \leq \text{addrstart} \leq 7FFFFFFF$
addrend	<p>Specifies in hexadecimal form the end address of the area up to which the specified TXT is output. For R-type, L-type and C-type members only.</p> $0 \leq \text{addrend} \leq 7FFFFFFF$

---

length	Specifies in hexadecimal form the length of the address area for the specified TXT. For R-type, L-type and C-type members only. $1 \leq \text{length} \leq 7\text{FFFFFFF}$
identl	Specifies the lower identification limit of the TXTP.
identu	Specifies the upper identification limit of the TXTP.
PHYSICAL	The physical LLM structure is listed.
LOGICAL	The logical LLM structure is listed.
	NEXT Only the next level down is listed.
	<u>ALL</u> The entire structure is listed.
?	The current value is logged.

The INFO processing operand affects LST.

## PAR KEY      Transfer file attributes and ISAM key

The KEY processing operand defines whether the file attributes and the ISAM key are to be included in the output member.

Operation	Processing operand
PAR	$K[KEY] = \left[ \begin{array}{c} \{ YES \} \\ \{ NO \} \\ \{ ? \} \end{array} \right]$

**YES**                      If PAR KEY = YES is specified for ADD, the file attributes such as filename, ACCESS-METHOD, RECORD-FORMAT, RECORD-SIZE, BUFFER-LENGTH, PADDING-FACTOR, LOGICAL-FLAG-LENGTH, VALUE-FLAG-LENGTH, PROPAGATE-VALUE-FLAG, USER-ACCESS and the ISAM key of each file are transferred to the output member, without any changes.

This specification is only permissible when processing program libraries, and is not permissible when processing source libraries.

**NO**                        The file attributes and ISAM keys are not transferred. In this case it is only possible to transfer ISAM files with KEY-POSITION=5, KEY-LENGTH ≤ 16 and RECORD-FORMAT=VARIABLE to the output member.

**?**                            The current value is logged.

### Notes

- The KEY=YES processing operand is not permitted when source libraries are processed.
- If a member is listed using LST and the member has stored ISAM keys, the KEY-POSITION and KEY-LENGTH values are also output.
- If a file with KEY-POSITION > 5 or RECORD-FORMAT=FIXED is added, the member containing this file cannot be edited or numbered.

The KEY processing operand affects ADD.

## PAR LCASE      Lowercase/uppercase conversion

Conventionally, LMS converts all terminal input to uppercase characters. Lowercase/uppercase conversion can be suppressed by means of the LCASE processing operand for COR and UPDR.

This operand is only effective with input from display terminals; it cannot be used for input from members or procedures.

Operation	Processing operand
PAR	LC[ASE] = { $\left. \begin{array}{c} \text{YES} \\ \text{NO} \\ ? \end{array} \right\}$ }

**YES**                      Lowercase letters are not converted to uppercase type.

**NO**                      All lowercase letters are converted to uppercase type.

**?**                          The current value is logged.

LCASE=YES is meaningful for operands whose operand values are allowed to consist of any string of characters (including lowercase letters) enclosed in apostrophes.

These operands are:

- the substatement \*CHANGE, for COR
- the substatements \*COR, \*DEL, \*ID, \*INV and \*REP, for UPDR.

When LCASE=YES is in force, all LMS statements, operands and operand values (except for those mentioned above) must be input as uppercase letters.

It is therefore advisable not to activate LCASE=YES until just before a correction and to reset it to LCASE=NO after the correction.

*Example*

```
/START-PROGRAM $LMS
$LIB LIBRARY,BOTH
$PAR LCASE=YES
$ADD >LETTER.A
*Dear ...
.
.
.
**END
$PAR LCASE=NO
$END
```

The text 'Dear ...' is stored exactly in this form in member LETTER.A (cf. ADD, format 4).

The LCASE processing operand affects the entire LMS run.



## PAR LINE Define number of lines and columns per log page

The LINE processing operand determines the length and width of a log page produced by LMS. The column specification for controlling the page width is interpreted for TOC, COM and LST. Processing operand LINE is only interpreted when the output medium is SYSLST or a member.

Operation	Processing operand
PAR	$\text{LIN[E]} = \left\{ \begin{array}{l} \text{[lines] [/columns]} \\ \underline{60/132} \\ ? \end{array} \right\}$

lines	Number of lines per page. $21 \leq \text{lines} \leq 255$  A new log page is started at the latest after "lines" lines (see PAR NEWFORM), and is always started whenever a change of function takes place.
columns	Number of columns per line. $21 \leq \text{columns} \leq 255$
?	The current value is logged.

The LINE processing operand affects:  
TOC, COM, LST

## PAR LOG      Log statements

The LOG processing operand controls the scope of the LMS log.

Operation	Processing operand
PAR	$LO[G] = \left\{ \begin{array}{l} \text{MAX} \\ \text{MED} \\ \underline{\text{MIN}} \\ ? \end{array} \right\}$

MAX	Complete log.
MED	Statements are logged only in the event of an error. Positive acknowledgments are logged.
<u>MIN</u>	Only error messages, end messages and negative acknowledgments are logged.
?	The current value is logged.

The LOG processing operand affects the entire LMS run and COR, NUM and UPD.

## PAR LST Define scope and mode of member listings

The LST processing operand is supported for reasons of compatibility only. Its functions have been assumed by the processing operands FORMAT and INFO.

The LST processing operand has two different formats:

### Format 1: Listing of text members

Operation	Operands
PAR	$LS[T] = \left[ \begin{array}{l} \left\{ \begin{array}{l} \underline{TXT} \\ REC \end{array} \right\} [ / \left\{ \begin{array}{l} NU [M] \\ NO [NUM] \end{array} \right\} ] \\ PRT \\ NO \\ ? \end{array} \right]$

### Format 2: Listing of object modules

Operation	Operands
PAR	$LS[T] = \left[ \begin{array}{l} REC [ / \left\{ \begin{array}{l} NU [M] \\ NO [NUM] \end{array} \right\} ] \\ ALL \\ \underline{TXT} \\ UPD \\ SYM \\ NO \\ ? \end{array} \right]$

The functions of processing operand LST are assumed by the processing operands FORMAT and INFO as shown in the following table:

LST value	FORMAT value	INFO value
REC	REC	ALL
ALL	SYM	ALL
TXT	SYM	(ESD, TXT, RLD)
UPD	SYM	(REP, TXTP)
SYM	SYM	(ESD, ISD, LSD)
NO	SYM	SUMMARY
PRT	P	ALL

**Format 1:** Listing of text members

This format applies only to member types S, M, J, P, D, X.  
It specifies the scope and mode of the listing process.

Operation	Processing operand
PAR	$LS [T] = \left[ \begin{array}{l} \left\{ \begin{array}{l} \underline{TXT} \\ \underline{REC} \end{array} \right\} [ / \left\{ \begin{array}{l} \underline{NU [M]} \\ \underline{NO [NUM]} \end{array} \right\} ] \\ \left\{ \begin{array}{l} \underline{PRT} \\ \underline{NO} \\ \underline{?} \end{array} \right\} \end{array} \right]$

- TXT                    Alphanumeric output.
- REC                    For each member record, two lines are output:
  1. Alphanumeric output
  2. Hexadecimal output with prefixed output of the 4-byte record length field
- PRT                    This specification applies only to list members (type P in program libraries, type S in source libraries). It has the same effect as the operand LAYOUT-CONTROL (CONTROL-CHARACTERS=EBCDIC) in the PRINT-FILE command (see the manual "User Commands (SDF Format)" [7]), i.e. the 1st character of the record contents is interpreted as a feed control character. LMS performs a validity check on this byte and replaces it with X'40' if it is not a valid feed control character. Valid feed control characters are all those that are listed as valid in the "Laser Printer" manual [9]. The user must ensure that the VFB (Vertical Format Buffer -- see the "Laser Printer" manual [9]) of the appropriate NDFILES contains all the control characters used in the member. On output, one member record corresponds to one list line.
- NO                      Only the number of member records is output.
- ?                        The current value is logged.
- NUM                    The record number of the member record is output before the record when the output mode is TXT; it is logged before the record length field when the output mode is REC.  
  
This option has no effect when PRT and NO are specified.
- NONUM                The record number is not output. This specification has no effect on output to SYSLST or to a member; in these cases the record number is always output.

**Format 2:** Listing of object modules

This format applies only to member type R. It specifies the scope and mode of the listing process.

Output from object modules can be subdivided into unedited output (REC) and edited output (ALL, TXT, UPD, SYM). In unedited output, the record length field and, if specified, the record number are output as well. In edited output, neither record length field nor record number are output.

Operation	Processing operand
PAR	$LS [T] = \left[ \begin{array}{l} REC [ / \left\{ \begin{array}{l} NU [M] \\ NO [NUM] \end{array} \right\} ] \\ ALL \\ \underline{TXT} \\ UPD \\ SYM \\ NO \\ ? \end{array} \right]$

- REC All records in the module are output in alphanumeric and hexadecimal form. The 4-byte record length field is logged before the hexadecimal line. The record number can be output (see NUM).
- ALL All ESD, ISD, TXT, RLD, TXTP, REP and END information available to LMS is output in edited form. Other information, e.g. LSD and DSDD, is output in unedited form. Continuous text information is not split up.
- TXT Lists TXT, ESD and RLD information in edited form. Continuous text is not split up into records.
- UPD The correction journal (TXTP) and REP information are output in edited form.
- SYM ESD and ISD information is output in edited form.
- NO The number of member records and the module length are output.
- ? The current value is logged.
- NUM The record number is output before the record length field when REC is specified. This option is only effective for output to the display terminal.
- NONUM The record number is not logged.

*Example*

```
/START-PROGRAM $LMS  
$LIB LMS.TEST,BOTH  
$PAR LST=REC/NUM  
$LSTR ERFAS
```

```
      E S D                      E R F A S          0  
#1  0054000B  02C5E2C44040404040001040400001C5D9C6C1E2404040F0000000000003  
  
      BA4040404040404040404040404040404040404040404040404040404040404040  
  
      4040404040404040404040404040404040404040404040404040404040404040  
      T X T                      &                &                &  
#2  00540014  02E3E7E3400000004040001040400001055041105066410000040A9C451050  
  
      1A4040404040404040404040404040404040404040404040404040404040404040  
  
      4040404040404040404040404040404040404040404040404040404040404040  
  
      .  
      .  
      .
```

Member ERFAS is listed in alphabetical and hexadecimal format, prefixed by a record number.



## PAR NEWFORM      Control form feed

The NEWFORM processing operand controls the form feed for the LMS log when the output medium is system file SYSLST or a member.

Operation	Processing operand
PAR	$N[EWFORM] = \left[ \begin{array}{l} \text{YES} \\ \text{NO} \\ \underline{3} \\ \text{number} \\ ? \end{array} \right]$

YES	Form feed to a new page either when processing operand LINE determines there should be a new page (see processing operand LINE, value "lines") or with each new statement or member.
NO	Form feed to a new page only when processing operand LINE determines there should be a new page.
number	Outputs "number" blank lines instead of performing a form feed at the start of output for a new element. "number" may have a value from 1 to 15. Default value: 3
?	The current value is logged.

### *Note*

If a page feed is to be performed after every member, PAR NEWFORM=YES must be specified. The default value 3 causes three blank lines to be output instead of a page feed.

Processing operand NEWFORM affects the entire LMS run.



## PAR OVERWRITE      Overwrite identically named members

The OVERWRITE processing operand controls the overwriting of identically named members in the output library, or of identically named files or FMS members in conjunction with SEL.

This operand is not valid for delta members.

Operation	Processing operand
PAR	$O[VERWRITE] = \left\{ \begin{array}{l} \text{YES} \\ \text{ONLY} \\ \text{NO} \\ \text{V} \\ \text{D} \\ \text{EXTEND} \\ \text{?} \end{array} \right\}$

- YES**                      An member or file with the same member designation is overwritten or, if it does not yet exist, it is created.
- ONLY**                    A member is written only if a member or file with the same member designation already exists.
- NO**                      A member or file with the same member designation is not overwritten and the statement is not executed.
- V**                         A member with the same member designation is overwritten only if version number new > version number old. This entry has the same effect as NO for program libraries.
- D**                         A member with the same member designation is overwritten only if date new > date old.
- EXTEND**                The member or the file is to be extended. This operand is only effective in conjunction with ADD and SEL. In the case of the other statements, EXTEND has the same effect as the operand NO.

A member or a file is, however, extended only if no ISAM keys are stored in the member and the file attributes stored in the member match the attributes of the file (apart from the file name). Otherwise ADD or SEL is rejected and an error message issued.
- ?**                         The current value is logged.

The OVERWRITE processing operand affects:  
 ADD, COR, DUP, EDR, EDT, NAM, NUM, PRT, SEL, UPD

## PAR PATH Specify a pathname

This processing operand is used to specify a pathname for UPDL and LSTL.

Operation	Processing operand
PAR	$PA[TH] = \left[ \begin{array}{c} \{name\} \\ ? \end{array} \right]$

**name** Pathname with a length of up to 255 characters

If this is specified with LSTL, only the sub-LLM with this pathname is listed.

If this is specified with UPDL, the name is used as the base for displacement in the \*COR substatement.

**?** The current value is logged.

The PATH processing operand affects UPDL and LSTL.

### Note

If the processing operand SLICE is set, it is reset to 'UNDEFINED' when the processing operand PATH is set.

## PAR PHASE      Define phase format

This processing operand defines the phase format to be generated. By default, PK phases are generated in the PAM key (PK) environment and NK phases in the non-PAM key (NK) environment. This processing operand can also be used to generate NK phases in the PK environment.

Operation	Processing operand
PAR	PH[ASE] = [ $\left\{ \begin{array}{l} \text{PK} \\ \text{NK} \\ ? \end{array} \right\}$ ]

PK            The phase is generated in PK format if it is to be written to a PK disk. If the phase is to be written to an NK disk, it is generated in NK format.

NK            The phase is always generated in NK format.

?             The current value is logged.

The default value of the PHASE operand is determined by the CLASS2-OPTION.

The PHASE processing operand affects SELC.

## PAR RANGE      Define check field in output records

The RANGE processing operand defines the position and length of the check field in output records. The check field can accommodate a value that is determined dynamically (operands STRING and VALUE). If input consists of an ISAM file, the ISAM key can be stored in the check field (processing operand STRING=KEY).

Operation	Processing operand
PAR	$RA[NGE] = \left\{ \begin{array}{l} [start] [/length] \\ \underline{NO} \\ ? \end{array} \right\}$

start	<p>Start position of the check field.  "start" may have the following values:  <math>1 \leq start \leq 251</math></p>
length	<p>Length of the check field.  "length" may have the following values:  <math>1 \leq length \leq 16</math></p> <p>If only "start" is specified, it must be in the range  <math>65 \leq start \leq 80</math></p> <p>This is because "length" is calculated as:  <math>length = 81 - start</math></p> <p>If only "length" is specified, "start" is calculated as:  <math>start = 81 - length</math></p> <p>If "start" and "length" are specified together, the following applies:  <math>start + length \leq 252</math></p>
<u>NO</u>	No check field is defined.
?	The current value is logged.

The RANGE processing operand affects:  
ADD, COR, EDR, EDT, NUM

## PAR REFERENCE Define reference conditions

The REFERENCE processing operand defines the reference conditions under which members are selected for processing. The reference condition consist of the pair: reference name and reference attribute. This processing operand is only evaluated for type R members in program libraries.

Operation	Processing operand
PAR	$RE[REFERENCE] = \left[ \begin{array}{l} \text{name} \\ \left( \left[ \text{name} \right], \left[ \begin{array}{l} \text{CSECT} \\ \text{ENTRY} \\ \text{ALL} \end{array} \right] \right) \\ ? \end{array} \right]$

- name** Specifies the reference name. Multiple selection is allowed. "name" may be up to 32 characters in length.
- CSECT** Only reference names with the CSECT attribute are to be processed.
- ENTRY** Only reference names with the ENTRY attribute are to be processed.
- ALL** Reference names with any attribute are to be processed.
- ?** The current value is logged.

The specification `PAR REFERENCE=` resets any existing reference conditions to "undefined".

`PAR REFERENCE=name` has the same effect as `PAR REFERENCE=(name, ALL)`.

`PAR REFERENCE=(, {...})` has the same effect as `PAR REFERENCE=(*, {...})`.

The REFERENCE processing operand affects:  
LST, TOC, DUP, DEL

*Note*

The member type R must be specified explicitly, otherwise the reference condition is ignored.

## PAR SEGMENT      Define segments of load module

The SEGMENT processing operand defines which segments of a load module are to be listed.

Operation	Processing operand
PAR	$SE[GMENT] = \left[ \begin{array}{l} \text{overlay} \\ \%ROOT \\ \underline{\%ALL} \\ ? \end{array} \right]$

overlay	Name of an overlay. "overlay" may be up to 8 characters in length.
%ROOT	Name of the root segment.
<u>%ALL</u>	All segments are listed.
?	The current value is logged.

The SEGMENT processing operand affects LST.

## PAR SLICE      Specify slice

This processing operand is used to specify a segment for UPDL and LSTL.

Operation	Processing operand
PAR	$SL[ICE] = \left[ \begin{array}{l} \{name\} \\ ? \end{array} \right]$

**name**      Slice name with a length of up to 32 characters

If this is specified with LSTL, only the slice with this name is listed.

If this is specified with UPDL, the name is used as the base for displacement in the \*COR substatement.

**?**            The current value is logged.

### *Note*

If the processing operand PATH is set, it is reset to 'UNDEFINED' when the processing operand SLICE is set.

## PAR SORT      Sort directory

The SORT processing operand defines the sort criteria for output of the directory entries (cf. TOC, page 162).

Operation	Processing operand
PAR	$SO[RT] = \left\{ \begin{array}{l} U \\ [N] [V] [D] \\ R \\ ? \end{array} \right\}$

- U**            Unsorted output:  
This parameter is only relevant for sequential libraries. Output follows the order of the members as they occur in the library.
- N**            Output is sorted by name.
- V**            Output is sorted by version number.
- D**            Output is sorted by date.
- R**            Output is sorted by reference name; these names have been defined using the REFERENCE processing operand.  
  
If no reference condition has been defined using the REFERENCE processing operand, LMS outputs the directory of primary names.
- ?**            The current value is logged.

N, V and D are the default values: they can be specified in any order and combination.

If more than one sort criterion is specified, sorting takes place in accordance with the order in which the criteria are specified.

The SORT processing operand affects TOC.



## PAR STRING Define string in check field of output records

The STRING processing operand defines a character string that is entered left-justified in the check field of the output records (controlled by RANGE) upon renumbering.

The KEY operand is supported for reasons of compatibility only. ISAM keys are added to the output member using the new processing operand PAR KEY=YES.

Operation	Processing operand
PAR	$\text{STRIN[G]} = \left[ \begin{array}{l} \text{'string'} \\ \text{NAME} \\ \text{KEY} \\ \text{NO} \\ \text{?} \end{array} \right]$

- string            Alphanumeric and special characters (up to 16 characters). The apostrophe is not allowed in the character string.
- NAME            The first 3 characters of the member name are adopted as the string.
- KEY             The KEY operand is supported for reasons of compatibility only. It stores the ISAM key in the check field, but it is ignored if the ISAM keys have been included in the output member by means of the PAR KEY=YES processing operand.  
  
If the lengths of the check field (see value "length" in the RANGE processing operand) and of the ISAM key (KEY-LENGTH) are not the same, the following rules apply:  
  
length>KEY-LENGTH: check field is padded on the right with zeros.  
length<KEY-LENGTH: ISAM key is truncated on the right.
- NO             No STRING entries are made in the check field.
- ?                The current value is logged.

The STRING processing operand affects:  
ADD, COR, EDR, EDT, NUM, SEL

## PAR STRIP Suppress records

The STRIP processing operand determines which record types are not to be transferred from the input member to the output member. STRIP is only interpreted in conjunction with members of the types R, L and C.

Operation	Processing operand
PAR	$\text{STRIP} = \left\{ \begin{array}{l} \text{rectype} \\ \text{rectype}, \dots \\ \text{YES} \\ \underline{\text{NO}} \\ ? \end{array} \right\}$

**rectype** The specified record type is excluded from transfer to the output member.

Possible record types are:

- ISD, LSD, TXTP, REP, INCLUDE, DSDD  
for R-type members
- TXTP  
for C-type and L-type members

Other specifications are ignored.

**YES** TXTP records are not transferred to the output member, i.e. the correction journal is not transferred.

**NO** All records are transferred to the output member.

**?** The current value is logged.

The STRIP processing operand affects UPD in the case of R-type, L-type and C-type members, and DUP in the case of R-type and C-type members.

## PAR SUM      Generate comparison statistics

The SUM processing operand controls the storage of comparison statistics resulting from a comparison specified by means of COM. These comparison statistics are placed in designated sum fields and can be processed further via SUM, SUMPRT, SUMDEL and SUMADD.

Operation	Processing operand
PAR	$\text{SUM} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \\ ? \end{array} \right\}$

**YES**                      The comparison statistics from subsequent comparisons are added to sum field S1.

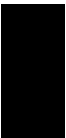
END has the following effect:

1. Sum field S1 is added to sum field S2.
2. Sum field S2 is output.
3. The LMS run is terminated.

NO                        The comparison statistics from subsequent comparisons are not stored.

The sum fields are not deleted at the time of their assignment. Deletion of the sum fields is effected by means of SUMDEL (see page 160).

The SUM processing operand affects COM.



## PAR TERMINATE Control termination procedure in error situations

The TERMINATE processing operand determines which cases are treated as errors and causes the internal termination code to be set.

If the termination code is set when LMS terminates, the program ends in a normal manner in interactive mode, but branches to STEP or ABEND or LOGOFF when in batch mode. Also in batch mode, LMS switches to test mode if serious errors occur, regardless of the current value.

This processing operand also controls the behavior of LMS after the occurrence of errors, i.e. it determines which errors cause LMS to switch to test mode.

Operation	Processing operand
PAR	$\text{TER}[\text{MINATE}] = \left[ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ ? \end{array} \right]$

- 1 Test mode      The termination bit is set in the event of serious errors, i.e. errors that would make continuation pointless (e.g. device errors).
- 2 Run mode, 3 Test mode  
The termination bit is set in the event of serious errors and also other errors (e.g. syntax error in statement, member could not be corrected).
- 4 Run mode, 5 Test mode  
As for 2 and 3, and also if a function cannot be executed because a member could not be found.
- 6 Run mode, 7 Test mode  
As for 4 and 5, and also if a function cannot be executed because overwriting of an existing member was not permitted (OVERWRITE=NO) or the member could not be written because there was not yet a member present with the same name (OVERWRITE=ONLY).
- ?                    The current value is logged.

With even values (2, 4, 6) the LMS run continues in run mode after the occurrence of the TERMINATE condition; with odd values (1, 3, 5, 7) LMS switches to test mode unless it is reading the statements in interactive mode from the display terminal.

The occurrence of each TERMINATE condition is logged.

The TERMINATE processing operand affects the entire LMS run.

## PAR TEST      Activate/deactivate and terminate test mode

The TEST processing operand activates test mode, regardless of any error conditions. It allows for a return to run mode, providing test mode was not caused by errors. Furthermore, it specifies that LMS terminates when run mode is switched to test mode.

Operation	Processing operand
PAR	$\text{TES [T]} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \\ \text{ABORT} \\ ? \end{array} \right\}$

YES	<p>LMS switches to test mode; statements other than END, LIB, PAR, SYS, CTL and PRT are not executed.</p> <p>The format of the statements is checked, as are the assignments of the required libraries. With the correction functions, the format of the correction statements is checked; also, all checks are performed in the member. Data records are skipped. Errors occurring during correction are logged.</p>
<u>NO</u>	LMS switches to run mode, i.e. all functions called are executed (default value).
ABORT	When run mode is switched to test mode on account of an error (see TERMINATE processing operand), LMS terminates.
?	The current value is logged.

If PAR TEST=YES is set, RST will have no effect.

The TEST processing operand affects the entire LMS run and RST.

## PAR TOC Control output format for directories of program libraries

The TOC processing operand determines the number of member entries per line and the way in which they are represented when the directory of a program library is output. The operand is interpreted together with the LINE processing operand.

Operation	Processing operand
PAR	$TO[C]=\left\{ \begin{array}{l} F \\ D \\ L \\ T \\ ? \end{array} \right\}$

- F** A member is entered for each log line. The maximum length is reserved for each type of entry (TYP=8, NAME=64, VERSION=24, VARIANT=4, DATE=10, FLAG=1). If the flag column contains a "D", the member is stored as a delta member (see Example 1). Processing operand LINE may be used to shorten the line to a maximum of 80 characters. If a member entry does not fit into the log line because its member name is too long, the line is split into two parts between member name and version.
- D** A complete tree of delta members is always listed, no matter which member of the tree has been specified in TOC. The member specification in TOC causes only the tree to be selected. As well as the member designation the internal delta number (DELTA#, reflects the chronological order) and the associated base number (BASE#) are issued. These internal delta numbers are unique within a tree, they define the chaining of the members in the tree (independent of the external user-own version designation). The output of a tree is always sorted by DELTA#, i.e., the SORT processing operand is not effective within a tree. Different trees are separated from each other by means of a continuous line.

The output fields DELTA# and BASE# for non-delta members are empty (see Example 2).
- L** The longest member entry in the entire directory is determined, and the number of possible member entries per log line is calculated on the basis of this figure. The LINE processing operand may also be used to reduce the number of characters per line. If member names are too long, the member entry is extended to 80 characters per line and, if the need arises, the line is split up between member name and version.

- I            The procedure in this case is similar to that for the value L. The longest member entry is determined separately for each member type. Depending on the length of the member name, varying numbers of member entries per log line can thus be generated for the various member types.
- ?
- The current value is logged.

The TOC processing operand affects TOC.

*Example 1*

TYP	NAME	VERSION	(VAR#)	DATE	FLAG
(S )	CTL	@	(0002)	1991-05-28	
(S )	DELTA-42	001	(0003)	1991-06-24	D
(S )	DELTA-42	002	(0003)	1991-06-23	D
(S )	DELTA-42	003	(0003)	1991-05-28	D

*Example 2*

TYP	NAME	VERSION	(VAR#)	DATE	DELTA#	BASE#
(S )	CTL		(0002)	1991-05-28		
(S )	DELTA-42	001	(0003)	1991-06-24	00001	00000
(S )	DELTA-42	002	(0003)	1991-06-23	00002	00001
(S )	DELTA-42	003	(0003)	1991-05-28	00003	00002



## PAR TYPE      Predefine member type

The TYPE processing operand predefines the member type, thus making it unnecessary to specify the member type explicitly in the statements.

Operation	Processing operand
PAR	$TY [PE] = \left[ \begin{array}{c} \{type\} \\ ? \end{array} \right]$

**type**                      Specifies the member type to be processed.

Valid member types are:

- S    Source programs
- M    Macros
- R    Object modules (not for delta members)
- C    Load modules (not for delta members)
- H    Compiler result information
- J    Procedures
- P    Edited data
- D    Text data
- X    Data of any format (not for delta members)
- \*    Stands for all member types (only for program libraries and DEL, DUP, LST, NAM, SEL, TOC)

**?**                              The current value is logged.

Specification of PAR TYPE= cancels any predefinition which may exist. The member type is then undefined.

The TYPE processing operand predefines the member type for:  
ADD, COM, COR, DEL, DUP, EDR, EDT, LST, NAM, NUM, SEL, TOC, UPD

## PAR VALUE Control numbering in check field of output records

The VALUE processing operand determines the initial value and the increment of the numbers which, during numbering, are entered left-justified in the check field of the output records, or in the ISAM key (if output is written to an ISAM file). Character strings defined by means of STRING are entered left-justified.

If STRING=KEY is specified and entry is from an ISAM file (see ADD), the processing operand VALUE will be ignored.

Operation	Processing operand
PAR	$V[ALUE] = \left\{ \begin{array}{l} [start] [/inc] \\ \underline{NO} \\ ? \end{array} \right\}$

start	Initial value for numbering (up to 16 characters). If this entry is omitted, numbering commences with 0.
inc	Increment (up to 16 characters). If this entry is omitted, numbering takes place in steps of 10.
<u>NO</u>	There is to be no numbering (default value).
?	The current value is logged.

If an overflow occurs during numbering (into the string portion, if one is present, or out of the check field), the overflowing digit is truncated and numbering is continued with the remainder after a warning message has been issued.

If this overflow occurs when output is written to an ISAM file, processing is aborted since the sequence of the ISAM key is no longer ascending (KEY OUT OF ORDER).

The VALUE processing operand affects:  
ADD, COR, EDR, EDT, NUM, SEL

## Examples

### Simple examples

#### Add, correct and assemble library source programs

A source program is added as an S-type member to a program library and then assembled. Since errors were found during assembly, the member is corrected with EDT and subsequently assembled again. The module from the EAM area is added to the same program library as an R-type member.

```

/START-PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX ----- (02)
PAR LOG=MAX
$LIB UEB.BIB,NEW,BOTH ----- (03)
LIB UEB.BIB,NEW,BOTH
LIBRARY IS CLEARED AND PREPARED ----- (04)
$ADDS QUELL.ERFASS>ERFASS ----- (05)
ADDS QUELL.ERFASS>ERFASS
INPUT FILE
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
      ADD QUELL.ERFASS AS (S)ERFASS/@(0001)/1991-07-25 ----- (06)
$TOC* * ----- (07)
TOC* *
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
TYP NAME VER (VAR#) DATE
(S) ERFASS @ (0001) 1991-07-25 ----- (08)
      1 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS }

```

## Examples

```
$LSTS ERFASS ----- (09)
LSTS ERFASS
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (S)ERFASS/@(0001)/1991-07-25
      TITLE 'DATA ENTRY'
      PRINT NOGEN
ERFAS      START
      BALR 5,0
      USING *,5
      OPEN DATEI,OUTPUT
LESEN      RDATA SATZ,ENDPGM
      PUT DATEI,SATZ
      B LESEN
ENDPGM      TERM
*
DATEI      FXB FCBTYPE=SAM, LINK=DATEN
SATZ      DS CL84
      END
NUMBER OF PROCESSED RECORDS IS 14
$END ----- (11)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/START-PROGRAM $ASSEMB ----- (12)
% BLS0500 PROGRAM 'ASSEMB', VERSION '300' OF '89-11-03' LOADED
V30.0A20 OF SIEMENS BS 2000 ASSEMBLER READY
GIVE ASSEMBLER OPTIONS !
**COMOPT SOURCE=UEB.BIB(ERFASS) ----- (13)
GIVE ASSEMBLER OPTIONS!
**END HALT
FLAGS IN 00003 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES ----- (14)
HIGHEST ERROR-WEIGHT : 1
SYSTEM MACROLIBRARY : :M:$TSOS.MACROLIB
SOURCE LIBRARY : :N:$USER.UEB.BIB
SOURCE PROGRAM : ERFASS
SOURCE VERS/DATE: @/910725
ASSEMBLY TIME : 1.5383 SEC.
/START-PROGRAM $LMS ----- (15)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX
PAR LOG=MAX
$LIB UEB.BIB,BOTH ----- (16)
LIB UEB.BIB,BOTH
```

**\$EDTS ERFASS** (17)  
EDTS ERFASS

```

0.10      TITLE 'DATA ENTRY'
0.20      PRINT NOGEN
0.30 ERFAS  START
0.40      BALR 5,0
0.50      USING *,5
0.60      OPEN  DATEI,OUTPUT
0.70 LESEN  RDATA SATZ,ENDPGM
0.80      PUT   DATEI,SATZ
0.90      B     LESEN
1.00 ENDPGM  TERM
1.10 *
x 1.20 DATEI  Fcb  FCBTYPE=SAM, LINK=DATEN
1.30 SATZ      DS    CL84
1.40 END
2.40
3.40
4.40
5.40
6.40
7.40
8.40
9.40

      OUTPUT ELEMENT= (S)ERFASS/@(0002)/1991-07-25
halt                                0000.10:001(0)

```

(18)

EDT0905 EDITED MEMBER TO BE ADDED? REPLY (Y=YES; N=NO) **y** (19)

```

INPUT  LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT  ELEMENT= (S)ERFASS/@(0001)/1991-07-25
OUTPUT ELEMENT= (S)ERFASS/@(0002)/1991-07-25
CORRECT (S)ERFASS/@(0001)/1991-07-25 AS (S)ERFASS/@(0002)/1991-07-25
      , OUTPUT REPLACED

```

(20)

**\$END**

END

% LMS0311 LMS V02.0A10 ENDED NORMALLY

**/DELETE-SYSTEM-FILE FILE-NAME=OMF**

**/ASSIGN-SYSDTA TO-FILE=\*LIBRARY-ELEMENT(LIBRARY=UEB.BIB,  
ELEMENT=ERFASS)**

**/START-PROGRAM \$ASSEMB**

% BLS0500 PROGRAM 'ASSEMB', VERSION '300' OF '89-11-03' LOADED  
V30.0A20 OF SIEMENS BS 2000 ASSEMBLER READY

GIVE ASSEMBLER OPTIONS !

FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES

HIGHEST ERROR-WEIGHT : -

SYSTEM MACROLIBRARY : :M:\$TSOS.MACROLIB

ASSEMBLY TIME : 2.1016 SEC.

**/START-PROGRAM \$LMS**

% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED

% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.

% ALL RIGHTS RESERVED

% LMS0310 LMS VERSION V02.0A10 LOADED

**\$PAR LOG=MAX**

PAR LOG=MAX

## Examples

---

```
$LIB UEB.BIB,BOTH (23)
LIB UEB.BIB,BOTH
$ADDR *OMF (24)
ADDR *OMF
INPUT OMF
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
      ADD ERFAS AS (R)ERFAS/(0001)/1991-07-25
$END (25)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/ASSIGN-SYSDTA TO-FILE=*PRIMARY (26)
/
```

- (01) LMS is called.
- (02) All messages and statements are logged.
- (03) Library UEB.BIB is to be created as a new program library and assigned as an I/O library.
- (04) Library UEB.BIB has been created.
- (05) File QUELL.ERFAS is added to the library as a type-S member having the name ERFAS.
- (06) Positive acknowledgment: member ERFAS, with maximum version designation @ and variant number 0001, is written to the library.
- (07) The directory of library UEB.BIB is to be listed.
- (08) Directory entry of library UEB.BIB.
- (09) Member ERFAS is to be listed.
- (10) Contents of member ERFAS.
- (11) LMS is terminated.
- (12) The assembler is called.
- (13) The source program in member ERFAS of program library UEB.BIB is to be assembled.
- (14) The program contains errors.
- (15) LMS is called again.
- (16) Program library UEB.BIB is assigned as the I/O library.
- (17) Member ERFAS is to be processed with EDT.

- (18) The error is corrected:
  - Make line 1.20 overwriteable by entering "x" in the statement column.
  - Change "FXB" in line 1.20 to "FCB" and terminate EDT by entering HALT in the statement line.
- (19) "Y" causes the corrected member to be added to the output library. Since the input library and output library are identical and no new name has been specified for the output member, the invalid input member is overwritten by the corrected output member.
- (20) Positive acknowledgment: input member ERFASS has been corrected. The output member is given the same name and the same version designation, the variant number is incremented by 1 to "0002".
- (21) The assembly run has been executed successfully.
- (22) LMS is called.
- (23) Program library UEB.BIB is reassigned as the I/O library.
- (24) Module ERFAS is taken from the EAM area and incorporated as member ERFAS.
- (25) LMS is terminated.
- (26) The system input file SYSDTA is reassigned.

Duplicate members

Members from a module library, macro library, source library and program library are duplicated to a program library.

```

/SET-FILE-LINK LINK-NAME=LIB001,FILE-NAME=MODUL.LIB }
/SET-FILE-LINK LINK-NAME=LIB002,FILE-NAME=MACRO.LIB }
/SET-FILE-LINK LINK-NAME=LIB003,FILE-NAME=QUELL.LIB } ----- (01)
/SET-FILE-LINK LINK-NAME=LIB004,FILE-NAME=TEST.LIB }
/START-PROGRAM $LMS ----- (02)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX
PAR LOG=MAX
$LIB LMSPL.LIB,NEW,BOTH ----- (03)
LIB LMSPL.LIB,NEW,BOTH
LIBRARY IS CLEARED AND PREPARED
$TOCR (1) ----- (04)
TOCR (1)
INPUT LIBRARY= :N:$USER.MODUL.LIB,LINK=LIB001,DEV=DISK
TYP NAME VER (VAR#) DATE
(R) MODERF @ (0001) 1991-07-26
1 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$DUPR MODERF(1)>MOD.ERF ----- (05)
DUPR MODERF(1)>MOD.ERF
INPUT LIBRARY= :N:$USER.MODUL.LIB,LINK=LIB001,DEV=DISK
OUTPUT LIBRARY= :N:$USER.LMSPL.LIB,DEV=DISK
DUPLICATE (R)MODERF/@(0001)/1991-07-26 AS (R)MOD.ERF/@(0001)/1991-07-2
$TOCM (2) ----- (06)
TOCM (2)
INPUT LIBRARY= :N:$USER.MACRO.LIB,LINK=LIB002,DEV=DISK
TYP NAME VER (VAR#) DATE NAME VER (VAR#) DATE
(M) MAC1 @ (0001) 1991-07-26 MAC2 @ (0001) 1991-07-26
2 (M)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$DUPM MAC*(2)>MU* ----- (07)
DUPM MAC*(2)>MU*
INPUT LIBRARY= :N:$USER.MACRO.LIB,LINK=LIB002,DEV=DISK
OUTPUT LIBRARY= :N:$USER.LMSPL.LIB,DEV=DISK
DUPLICATE (M)MAC1/@(0001)/1991-07-26 AS (M)MUC1/@(0001)/1991-07-26 }
DUPLICATE (M)MAC2/@(0001)/1991-07-26 AS (M)MUC2/@(0001)/1991-07-26 }----- (08)
$TOCS (3) ----- (09)
TOCS (3)
INPUT LIBRARY= :N:$USER.QUELL.LIB,LINK=LIB003,DEV=DISK
TYP NAME VER (VAR#) DATE NAME VER (VAR#) DATE
(S) EDTB @ (0001) 1991-07-26 PROT @ (0001) 1991-07-26
(S) SEINAUS @ (0001) 1991-07-26 SERFAS @ (0001) 1991-07-26
4 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$DUPS *,-EDTB(3) ----- (10)
DUPS *,-EDTB(3)
INPUT LIBRARY= :N:$USER.QUELL.LIB,LINK=LIB003,DEV=DISK
OUTPUT LIBRARY= :N:$USER.LMSPL.LIB,DEV=DISK
DUPLICATE (S)PROT/@(0001)/1991-07-26 AS (S)PROT/@(0001)/1991-07-26
DUPLICATE (S)SEINAUS/@(0001)/1991-07-26 AS
(S)SEINAUS/@(0001)/1991-07-26
DUPLICATE (S)SERFAS/@(0001)/1991-07-26 AS (S)SERFAS/@(0001)/1991-07-26

```



```

$TOC* (4) (11)
TOC* (4)
INPUT LIBRARY= :N:$USER.TEST.LIB, LINK=LIB004, DEV=DISK
TYP NAME VER (VAR#) DATE
(S) SERFAS @ (0001) 1991-07-26
1 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$DUP* (4)>*/007 (12)
DUP* (4)>*/007
INPUT LIBRARY= :N:$USER.TEST.LIB, LINK=LIB004, DEV=DISK
OUTPUT LIBRARY= :N:$USER.LMSPL.LIB, DEV=DISK
DUPLICATE (S)SERFAS/@(0001)/1991-07-26
(S)SERFAS/007(0001)/1991-07-26
$TOC* * (13)
TOC* *
INPUT LIBRARY= :N:$USER.LMSPL.LIB, DEV=DISK
TYP NAME VER (VAR#) DATE NAME VER (VAR#) DATE
(M) MUC1 @ (0001) 1991-07-26 MUC2 @ (0001) 1991-07-26
2 (M)-ELEMENT(S) IN THIS TABLE OF CONTENTS
TYP NAME VER (VAR#) DATE
(R) MOD.ERF @ (0001) 1991-07-26
1 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS
TYP NAME VER (VAR#) DATE NAME VER (VAR#) DATE
(S) PROT @ (0001) 1991-07-26 SEINAUS @ (0001) 1991-07-26
(S) SERFAS @ (0001) 1991-07-26
3 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$TOC* */* (14)
TOC* */*
INPUT LIBRARY= :N:$USER.LMSPL.LIB, DEV=DISK
TYP NAME VER (VAR#) DATE NAME VER (VAR#) DATE
(M) MUC1 @ (0001) 1991-07-26 MUC2 @ (0001) 1991-07-26
2 (M)-ELEMENT(S) IN THIS TABLE OF CONTENTS
TYP NAME VER (VAR#) DATE
(R) MOD.ERF @ (0001) 1991-07-26
1 (R)-ELEMENT(S) IN THIS TABLE OF CONTENTS
TYP NAME VER (VAR#) DATE NAME VER (VAR#) DATE
(S) PROT @ (0001) 1991-07-26 SEINAUS @ (0001) 1991-07-26
(S) SERFAS 007 (0001) 1991-07-26 SERFAS @ (0001) 1991-07-26
4 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$LIB ? (15)
LIB ?
USAGE STATUS FORMAT LID LINKNAME FILENAME
IN OPEN PL :N:$USER.LMSPL.LIB
OUT OPEN PL :N:$USER.LMSPL.LIB
CLOSED PL 001 LIB001 :N:$USER.MODUL.LIB
CLOSED PL 002 LIB002 :N:$USER.MACRO.LIB
CLOSED PL 003 LIB003 :N:$USER.QUELL.LIB
CLOSED PL 004 LIB004 :N:$USER.TEST.LIB
$LIB C (16)
LIB C
$LIB ? (17)
LIB ?
USAGE STATUS FORMAT LID LINKNAME FILENAME
CLOSED PL :N:$USER.LMSPL.LIB
CLOSED PL 001 LIB001 :N:$USER.MODUL.LIB
CLOSED PL 002 LIB002 :N:$USER.MACRO.LIB
CLOSED PL 003 LIB003 :N:$USER.QUELL.LIB
CLOSED PL 004 LIB004 :N:$USER.TEST.LIB

```

## Examples

---

```
$END _____ (18)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/
```

- (01) Libraries assigned in this manner can be addressed in the LMS run using their short designation. In this example they are used as temporary input libraries.
- (02) LMS is invoked.
- (03) The new program library LMSPL.LIB is to be created, and to be assigned as the I/O library.
- (04) The directory of object module library MODUL.LIB, assigned by way of the short library designation (1), is to be listed.
- (05) Module MODERF from library MODUL.LIB is duplicated to library LMSPL.LIB under the member name MOD.ERF.
- (06) The directory of macro library LIB.MAC, assigned by way of the short library designation (2), is to be listed.
- (07) Those members of library LIB.MAC whose member designations begin with "MAC" are to be duplicated to library LMSPL.LIB. The new member designations begin with "MU". The member designations of the input members are transferred, starting from the third position.
- (08) Positive acknowledgment: the selected members, with the new member designations, are duplicated from the module library to the program library.
- (09) The directory of source library QUELL.LIB, assigned by way of the short library designation (3), is to be listed.
- (10) All members from source library QUELL.LIB, with the exception of member EDTB, are duplicated to the output library.
- (11) The directory of program library TEST.LIB, assigned by way of the short library designation (4), is to be listed.
- (12) All members of program library TEST.LIB are duplicated to the output library and stored under the same name and version number 007.
- (13) The directory of the current input library LMSPL.LIB is to be listed.
- (14) The directory of the current input library LMSPL.LIB is to be listed with all members. There are two members with member name SERFAS, but with different version numbers. TOC\* \* only indicated the member with the highest version number, "SERFAS/007".

- (15) The status of the libraries used during the LMS run is queried.
- (16) Library LMSPL.LIB is closed.
- (17) The status of the libraries used during the LMS run is queried.
- (18) The LMS run is terminated.



## Examples

---

### Compare members

Member ERFASS (listed in the example on page 253) and member EINAUS are compared. A comparison log is generated.

```
/START-PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX
PAR LOG=MAX
$LIB UEB.BIB,BOTH ----- (02)
LIB UEB.BIB,BOTH
$ADDS QUELL.EINAUS>EINAUS ----- (03)
ADDS QUELL.EINAUS>EINAUS
INPUT FILE
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
      ADD QUELL.EINAUS AS (S)EINAUS/@(0001)/1991-07-29
$LSTS EINAUS ----- (04)
LSTS EINAUS
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (S)EINAUS/@(0001)/1991-07-29
      TITLE 'DATA ENTRY'
      PRINT NOGEN
ERFAS  START
      BALR 5,0
      USING *,5
      OPEN DATEI,OUTPUT
LESEN  RDATA SATZ,ENDPGM
      CLC TEXT(4),=C'/EOF'
      BE  ENDPGM
      MVC ATEXT,TEXT
      LH  9,SL
      AH  9,=H'1'
      STH 9,ASL
      WROUT ASATZ,ENDPGM
      PUT DATEI,SATZ
      B  LESEN
ENDPGM TERM
*
DATEI  FCB  FCBTYPE=SAM,LINK=DATEN
      DS  OH
SATZ   DS  CL84
SL     DS  CL2
      DS  CL2
TEXT   DS  CL80
ASATZ  DS  OCL85
ASL    DS  CL2
      DC  X'000001'
ATEXT  DS  CL80
      END
NUMBER OF PROCESSED RECORDS IS      29
$PAR COMPARE=/MAX ----- (05)
PAR COMPARE=/MAX
$COMS EINAUS=ERFASS ----- (06)
COMS EINAUS=ERFASS
```

```

FUNCTION          = C O M P A R E
PAR              COMPARE= 00001/00072/L/MAX
PRIMARY         LIBRARY= :N:$USER.UEB.BIB
PRIMARY         ELEMENT= (S)EINAUS/@(0001)/1991-07-29
SECONDARY      LIBRARY= :N:$USER.UEB.BIB
SECONDARY      ELEMENT= (S)ERFASS/@(0002)/1991-07-26
    
```

```

SAME FROM      #1 TO      #7 AS FROM      #1 TO      #7
    
```

```

#1 >          TITLE 'DATA ENTRY'<
#2 >          PRINT NOGEN<
#3 >ERFAS     START<
#4 >          BALR 5,0<
#5 >          USING *,5<
#6 >          OPEN  DATEI,OUTPUT<
#7 >LESEN     RDATA SATZ,ENDPGM<
    
```

```

INS. FROM      #8 TO      #14
    
```

```

#8 >          CLC   TEXT(4),=C'/EOF'<
#9 >          BE   ENDPGM<
#10 >         MVC  ATEXT,TEXT<
#11 >         LH   9,SL<
#12 >         AH   9,=H'1'<
#13 >         STH  9,ASL<
#14 >         WROUT ASATZ,ENDPGM<
    
```

```

SAME FROM      #15 TO     #19 AS FROM      #8 TO      #12
    
```

```

#15 >         PUT  DATEI,SATZ<
#16 >         B   LESEN<
#17 >ENDPGM   TERM<
#18 >*<
#19 >DATEI     FCB  FCBTYPE=SAM, LINK=DATEN<
    
```

```

INS.           #20
    
```

```

#20 >         DS   OH<
    
```

```

SAME           #21          AS           #13
    
```

```

#21 >SATZ     DS   CL84<
    
```

```

INS. FROM      #22 TO     #28
    
```

```

#22 >SL       DS   CL2<
#23 >         DS   CL2<
#24 >TEXT     DS   CL80<
#25 >ASATZ   DS   OCL85<
#26 >ASL     DS   CL2<
#27 >         DC   X'000001'<
#28 >ATEXT   DS   CL80<
    
```

```

SAME           #29          AS           #14
    
```

```

#29 >         END<
    
```

```

PRIMARY      ELEMENT= (S)EINAUS/@(0001)/1991-07-29
    
```

## Examples

---

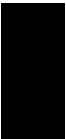
```
SECONDARY ELEMENT= (S)ERFASS/@(0002)/1991-07-26
RESULT: C   PRIMARY= 29 INSERTED= 15 ( 3) DELETED= 0 ( 0) } (22)
          SECONDARY= 14      SAME= 14 ( 4) } (23)
$END
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/
```

- (01) LMS is invoked.
- (02) Program library UEB.BIB is assigned as the input/output library.
- (03) File QUELL.EINAUS is added to the library as type-S member EINAUS.
- (04) Member EINAUS is listed.
- (05) The COMPARE processing operand defines that the comparison log is to be output in its entirety.
- (06) Members EINAUS and ERFASS are compared.
- (07) Start of comparison log:  
The log includes the values set for the COMPARE processing operand, the names of the primary and secondary libraries, and the names of the primary and secondary members.
- (08) - (21)  
Comparison log
- (08) The records with record IDs #1 through #7 are identical in both members.
- (09) Output of identical records.
- (10) Records #8 through #14 are present in the primary member only and are represented as INS(erted).
- (11) Output of inserted records.
- (12) Records #15 through #19 of the primary member are identical to records #8 through #12 of the secondary member.
- (13) Output of identical records.
- (14) Record #20 is present in the primary member only and is represented as INS(erted).
- (15) Output of inserted record.
- (16) Record #21 of the primary member is identical to record #13 of the secondary member.

- (17) Output of identical record.
- (18) Records #20 through #28 are present in the primary member only and are represented as INS(erted).
- (19) Output of inserted records.
- (20) Record #29 of the primary member is identical to record #14 of the secondary member.
- (21) Output of identical record.
- (22) Result of the comparison; output of the number of records of the primary and secondary members, and of the number of inserted, identical and deleted records.

The numbers in parentheses indicate how many continuous sections (consisting of consecutive records) have been inserted, identified as identical, or deleted.

- (23) LMS is terminated.



## Examples

---

### Processing delta members

```
/START-PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX
PAR LOG=MAX
$LIB LIB.DELTA,NEW,BOTH ----- (02)
LIB LIB.DELTA,NEW,BOTH
LIBRARY IS CLEARED AND PREPARED
$ADDS WORKELEM>DELTA/V00,BASEVERSION=*NONE ----- (03)
ADDS WORKELEM>DELTA/V00,BASEVERSION=*NONE
INPUT FILE
OUTPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
      ADD WORKELEM AS (S)DELTA/V00(0001)/1991-07-29 , FIRST DELTA VERSION
$EDTS DELTA/V00>VOLLELEM ----- (04)
EDTS DELTA/V00>VOLLELEM
```

```
0.10  MINI      START
0.20          BALR  3,0
0.30          USING *,3
0.40          OPEN  SAMFCB,OUTPUT
0.50          PUT   SAMFCB,EINGABE
0.60          TERM
0.70  SAMFCB   FCB   FCBTYP=SAM,LINK=MINI
0.80  EINGABE DC   C'THERE YOU ARE'
0.90          END
1.90
2.90
3.90
4.90
5.90
6.90
7.90
8.90
9.90
10.90
11.90
12.90
13.90
                                OUTPUT ELEMENT= (S)VOLLELEM/V00(0001)/1991-07-29
halt                                0000.10:001(0)
```

```
EDT0905 EDITED MEMBER TO BE ADDED? REPLY (Y=YES; N=NO) y
INPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
OUTPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
INPUT ELEMENT= (S)DELTA/V00(0001)/1991-07-29
OUTPUT ELEMENT= (S)VOLLELEM/V00(0001)/1991-07-29
      CORRECT (S)DELTA/V00(0001)/1991-07-29 AS
      (S)VOLLELEM/V00(0001)/1991-07-29
```



```

$DUPS VOLLELEM>DELTA/V01,BASEVERSION=*HIGH ----- (05)
DUPS VOLLELEM>DELTA/V01,BASEVERSION=*HIGH
INPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
OUTPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
      DUPLICATE (S)VOLLELEM/V00(0001)/1991-07-29 AS
              (S)DELTA/V01(0002)/1991-07-29 ON BASE
              (S)DELTA/V00(0002)/1991-07-29

$LIB LIB.ARBEIT,IN ----- (06)
LIB LIB.ARBEIT,IN
$DUPS INPUT>DELTA/V02,BASEVERSION=V00 ----- (07)
DUPS INPUT>DELTA/V02,BASEVERSION=V00
INPUT LIBRARY= :N:$USER.LIB.ARBEIT,DEV=DISK
OUTPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
      DUPLICATE (S)INPUT/@(0001)/1991-07-29 AS (S)DELTA/V02(0003)/1991-07-29
              ON BASE (S)DELTA/V00(0003)/1991-07-29

$LIB LIB.DELTA,IN ----- (08)
LIB LIB.DELTA,IN
$PAR TOC=D ----- (09)
PAR TOC=D
$TOC ----- (10)
$TOC* */* GENERATED BY LMS
INPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
TYP  NAME          VERSION      (VAR#)  DATE          DELTA#  BASE#
(S   ) DELTA . . . . . V00 . . . (0003)  1991-07-29  00001  00000
(S   ) DELTA . . . . . V01 . . . (0003)  1991-07-29  00002  00001
(S   ) DELTA . . . . . V02 . . . (0003)  1991-07-29  00003  00001
-----
(S   ) VOLLELEM . . . . . V00 . . . (0001)  1991-07-29
-----
4 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$ADDS WORKELEM>DELTA/V11,BASEVERSION=V01 ----- (11)
ADDS WORKELEM>DELTA/V11,BASEVERSION=V01
INPUT FILE
OUTPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
      ADD WORKELEM AS (S)DELTA/V11(0004)/1991-07-29 ON BASE
      (S)DELTA/V01(0004)/1991-07-29

$TOC ----- (12)
$TOC* */* GENERATED BY LMS
INPUT LIBRARY= :N:$USER.LIB.DELTA,DEV=DISK
TYP  NAME          VERSION      (VAR#)  DATE          DELTA#  BASE#
(S   ) DELTA . . . . . V00 . . . (0004)  1991-07-29  00001  00000
(S   ) DELTA . . . . . V01 . . . (0004)  1991-07-29  00002  00001
(S   ) DELTA . . . . . V02 . . . (0004)  1991-07-29  00003  00001
(S   ) DELTA . . . . . V11 . . . (0004)  1991-07-29  00004  00002
-----
(S   ) VOLLELEM . . . . . V00 . . . (0001)  1991-07-29
-----
5 (S)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$END ----- (13)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/

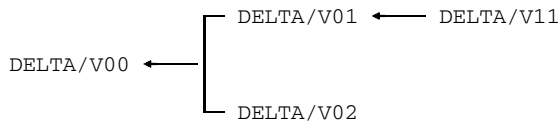
```

## Examples

---

- (01) LMS is invoked.
- (02) Program library LIB.DELTA is created and assigned as the input/output library.
- (03) File WORKELEM is added to the library as a new type-S delta member, DELTA/V00.
- (04) Delta member DELTA/V00 is to be processed with EDT. To do so, a new non-delta member, VOLLELEM, is generated from delta member DELTA/V00.
- (05) The processed non-delta member, VOLLELEM, is duplicated as type-S delta member DELTA/V01 to delta member DELTA/V00 in the output library.
- (06) Program library LIB.ARBEIT is assigned as the input library.
- (07) Non-delta member INPUT is duplicated as type-S delta member DELTA/V02 to delta member DELTA/V00 in the output library.
- (08) Program library LIB.DELTA is reassigned as the input library.
- (09) Processing operand PAR TOC=D is mandatory for the complete delta tree to be listed when TOC is called.
- (10) The directory of library LIB.DELTA is to be listed.
- (11) File WORKELEM is added as type-S delta member DELTA/V11 to delta member DELTA/V01.
- (12) The directory of library LIB.DELTA is to be listed.
- (13) LMS is terminated.

The delta members have the following structure:



## Complex examples

### Correct a source program using COR

Member DAT is corrected with the aid of COR correction statements.

```

/START-PROGRAM $LMS _____ (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX!LIB UEB.BIB,BOTH _____ (02)
PAR LOG=MAX
LIB UEB.BIB,BOTH
$ADDS QUELL.DAT>DAT _____ (03)
ADDS QUELL.DAT>DAT
INPUT FILE
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
ADD QUELL.DAT AS (S)DAT/@(0001)/1991-07-30
$PAR LST=TXT/NUM _____ (04)
PAR LST=TXT/NUM
$LSTS DAT _____ (05)
LSTS DAT
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (S)DAT/@(0001)/1991-07-30
#1 >TEST START
#2 >TEST START
#3 > BALR 3.0
#4 > USING *,3
#5 > GDATE DATUM,FORMAT=ISO
#6 > WROUT SATZ1,ENDE
#7 >ENDE TERM
#8 >*
#9 > DS OF
#10 >SATZ1 DC AL2(17)
#11 > DC X'000001'
#12 >DATUM DS CL12
#13 > END TEST
NUMBER OF PROCESSED RECORDS IS 13
$DUPS DAT>SDAT _____ (07)
DUPS DAT>SDAT
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
DUPLICATE (S)DAT/@(0001)/1991-07-30 AS (S)SDAT/@(0001)/1991-07-30
$CORS DAT _____ (08)
CORS DAT
**DEL #1 _____ (09)
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (S)DAT/@(0001)/1991-07-30
OUTPUT ELEMENT= (S)DAT/@(0002)/1991-07-30
*DEL #1
*DEL* #1 >TEST START

```

## Examples

```

**CHA #3'.0'<17>:=',0' _____ (10)
*CHA #3'.0'<17>:=',0'
      #1 >TEST      START
**REP #5 _____ (11)
*REP #5
*HIT*      #2 >      BALR  3,0
           #3 >      USING *,3
*DEL*      #5 >      GDATE DATUM,FORMAT=ISO
           GDATE DATUM,FORMAT=ISO,TOD=ZEIT _____ (12)
*ADD*      #4 >      GDATE DATUM,FORMAT=ISO,TOD=ZEIT
**INS #6 _____ (13)
*INS #6
           #5 >      WROUT SATZ1,ENDE
*          WROUT SATZ2,ENDE
*ADD*      #6 >      WROUT SATZ2,ENDE
**INS #12 _____ (14)
*INS #12
           #7 >ENDE   TERM
           #8 >*
           #9 >      DS    OF
           #10 >SATZ1 DC    AL2(17)
           #11 >      DC    X'000001'
           #12 >DATUM DS    CL12
*SATZ2   DC    A(13)
*ADD*      #13 >SATZ2 DC    A(13)
*          DC    X'000001'
*ADD*      #14 >      DC    X'000001'
*ZEIT    DS    CL8
*ADD*      #15 >ZEIT  DS    CL8
**END _____ (15)
*END
           #16 >      END    TEST
CORRECT (S)DAT/@(0001)/1991-07-30 AS (S)DAT/@(0002)/1991-07-30
, OUTPUT REPLACED
$LSTS DAT _____ (16)
LSTS DAT
INPUT  LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT  ELEMENT= (S)DAT/@(0002)/1991-07-30
      #1 >TEST      START
      #2 >          BALR  3,0
      #3 >          USING *,3
      #4 >          GDATE DATUM,FORMAT=ISO,TOD=ZEIT
      #5 >          WROUT SATZ1,ENDE
      #6 >          WROUT SATZ2,ENDE
      #7 >ENDE      TERM
      #8 >*
      #9 >          DS    OF
      #10 >SATZ1    DC    AL2(17)
      #11 >          DC    X'000001'
      #12 >DATUM    DS    CL12
      #13 >SATZ2    DC    A(13)
      #14 >          DC    X'000001'
      #15 >ZEIT     DS    CL8
      #16 >          END    TEST
NUMBER OF PROCESSED RECORDS IS      16

```

---

```
$END _____ (17)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/
```

- (01) LMS is invoked.
- (02) Two statements separated by an exclamation mark are entered:
  - a) All messages and statements are to be logged.
  - b) Program library UEB.BIB is assigned as the input/output library.
- (03) File QUELL.DAT is incorporated in the library as an S-type member having the name DAT.
- (04) Processing operand LST defines that the record number is to be output as well during the listing procedure.
- (05) Member DAT is listed.
- (06) Contents of member DAT with prefixed record number.
- (07) Member DAT is duplicated into member SDAT.
- (08) Member DAT is to be corrected with COR.
- (09) The record with record number #1 is to be deleted.
- (10) Character string '.0' in column 17 of the record with number #3 is changed into '0'.
- (11), (12)  
The record with number #5 is replaced by that denoted by (12).
- (13), (14)  
Subsequent to the records with numbers #6 and #12, records are inserted.
- (15) End of correction statements.
- (16) The corrected member is listed once more.
- (17) LMS is terminated.

Correct an object module using UPD

Member USELST is corrected with the aid of UPDR correction statements. The cross control number is first computed in test mode and then entered for checking purposes during correction.

```
/START-PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX,TEST=YES ----- (02)
PAR LOG=MAX,TEST=YES
$LIB UEB.BIB,BOTH ----- (03)
LIB UEB.BIB,BOTH
$PAR LCASE=YES ----- (04)
PAR LCASE=YES
$UPDR USELST ----- (05)
UPDR USELST
**COR C0,'ER'::='aa' ----- (06)
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (R)USELST/@(0001)/1991-07-31
*COR C0,'ER'::='aa'
**END ----- (07)
*END
*COR C0,'ER'::='aa'
CONTROL NUMBER: 009926
TEXT-ADR: 000000C0
TEXT BEFORE CHANGE: E R
TEXT AFTER CHANGE: a a
8181
CROSS CONTROL NUMBER:00009926 } ----- (08)
NO CORRECT (R)USELST/@(0001)/1991-07-31 , LMS IN TESTMODE !!! ----- (09)
$PAR TEST=NO ----- (10)
PAR TEST=NO
$UPDR USELST ----- (11)
UPDR USELST
**CON 9926 ----- (12)
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (R)USELST/@(0001)/1991-07-31
OUTPUT ELEMENT= (R)USELST/@(0002)/1991-07-31
*CON 9926
**COR C0,'ER'::='aa' ----- (13)
*COR C0,'ER'::='aa'
**END
*END
*CON 9926
*COR C0,'ER'::='aa'
CONTROL NUMBER: 009926
TEXT-ADR: 000000C0
TEXT BEFORE CHANGE: E R
TEXT AFTER CHANGE: a a
8181
```

```
CROSS CONTROL NUMBER:00009926
CORRECT (R)USELST/@(0001)/1991-07-31 AS (R)USELST/@(0002)/1991-07-31
, OUTPUT REPLACED (14)
$PAR LCASE=NO (15)
PAR LCASE=NO
$END (16)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/
```

- (01) LMS is invoked.
- (02) Processing operands LOG and TEST are specified via PAR:
  - a) All messages and statements are logged.
  - b) Test mode is activated.
- (03) Program library UEB.BIB is assigned as the input/output library.
- (04) Lower case letters entered are not converted to upper case.
- (05) Module USELST is to be corrected.
- (06) The text of address 0000C0 is replaced.
- (07) End of correction statements.
- (08) LMS checks the correction specifications and computes the control number and the cross control number.
- (09) The corrections are not performed because test mode is activated.
- (10) Test mode is deactivated.
- (11) Module USELST is to be corrected.
- (12) The cross control number is entered.
- (13) The text of address 0000C0 is replaced.
- (14) The corrections have been performed.
- (15) All entries are converted to upper case.
- (16) LMS is terminated.

Compare members and prepare correction statements

Correction statements are issued when the members DAT and SDAT are compared. These are written to a file and again incorporated as members. Members DAT and SDAT are listed in the example on page 269.

```

/START=PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX,COM=/MAX/COR,SUM=YES ----- (02)
PAR LOG=MAX,COM=/MAX/COR,SUM=YES
$SYS ASSIGN-SYSOPT TO-FILE=COR.ELEM ----- (03)
SYS ASSIGN-SYSOPT TO-FILE=COR.ELEM
$LIB UEB.BIB,BOTH ----- (04)
LIB UEB.BIB,BOTH
$COMS DAT=SDAT ----- (05)
COMS DAT=SDAT
FUNCTION          = C O M P A R E
PAR              COMPARE= 00001/00072/L/MAX/COR      COR-CARD OUTPUT=SYSOPT
PRIMARY          LIBRARY= :N:$USER.UEB.BIB
PRIMARY          ELEMENT= (S)DAT/@(0002)/1991-08-01
SECONDARY        LIBRARY= :N:$USER.UEB.BIB
SECONDARY        ELEMENT= (S)SDAT/@(0001)/1991-08-01
-----
SAME              #1              AS              #1
                  #1 >TEST        START<
-----
DEL.              FROM           #2 TO           #3
                  #2 >TEST        START<
                  #3 >            BALR  3,0<
-----
INS.              #2
                  #2 >            BALR  3,0<
-----
SAME              #3              AS              #4
                  #3 >            USING *,3<
-----
DEL.              #5
                  #5 >            GDATE DATUM,FORMAT=ISO<
-----
INS.              #4
                  #4 >            GDATE DATUM,FORMAT=ISO,TOD=ZEIT<
-----
SAME              #5              AS              #6
                  #5 >            WROUT SATZ1,ENDE<
-----
INS.              #6

```



```

#6 >          WROUT SATZ2, ENDE<
SAME FROM      #7 TO      #12 AS FROM      #7 TO      #12

#7 > ENDE      TERM<
#8 > * <
#9 >          DS      OF<
#10 > SATZ1    DC      AL2(17)<
#11 >          DC      X'000001'<
#12 > DATUM    DS      CL12<

INS. FROM      #13 TO      #15

#13 > SATZ2    DC      A(13)<
#14 >          DC      X'000001'<
#15 > ZEIT     DS      CL8<

SAME           #16          AS           #13

#16 >          END      TEST<

PRIMARY        ELEMENT= (S)DAT/@(0002)/1991-08-01
SECONDARY      ELEMENT= (S)SDAT/@(0001)/1991-08-01
RESULT: C      PRIMARY=    16  INSERTED=    6 (    4)  DELETED=    3 (    2)
              SECONDARY=   13    SAME=    10 (    5)

$SUM                                                    (06)
SUM
AREA C1
          PRIM.   PRIM.   INS.   SAME   DEL.   INS+DEL   SEC.   SEC.
STATISTIC  ELEM.   LINES  LINES  LINES  LINES   LINES  ELEM.
S (SAME)   0       0      -     0      -       -     0     0
C (CHANGED) 1       16     6     10     3       9     13    1
I (INSERTED) 0       0      0     -     -       0     -     -
D (DELETED) -       -      -     -     0       0     0     0

          TOTAL   1       16     6     10     3       9     13    1
$SYS ASSIGN-SYSOPT TO-FILE=*PRIMARY                    (07)
SYS ASSIGN-SYSOPT TO-FILE=*PRIMARY
$ADDJ COR.ELEM                                         (08)
ADDJ COR.ELEM
INPUT FILE
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
              ADD COR.ELEM AS (J)COR.ELEM/@(0001)/1991-08-01
$LSTJ COR.ELEM                                         (09)
LSTJ COR.ELEM
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (J)COR.ELEM/@(0001)/1991-08-01
$PAR CHECK=NO
$CORS SDAT/@/1991-08-01
*DEL #2-#3
*INS #3
          BALR  3,0
*DEL #5
*INS #5
          GDATE DATUM,FORMAT=ISO,TOD=ZEIT
*INS #6
          WROUT SATZ2, ENDE
*INS #12

```

## Examples

---

```
SATZ2    DC    A(13)
          DC    X'000001'
ZEIT     DS    CL8
*END
NUMBER OF PROCESSED RECORDS IS      15
$END                                          (10)
END
AREA C2
          PRIM.  PRIM.  INS.  SAME  DEL.  INS+DEL  SEC.  SEC.
STATISTIC ELEM.  LINES  LINES  LINES  LINES  LINES  LINES  ELEM.
S (SAME)      0      0      -      0      -      -      0      0
C (CHANGED)   1     16      6     10      3      9     13     1
I (INSERTED)  0      0      0      -      -      0      -      -
D (DELETED)   -      -      -      -      0      0      0      0

          TOTAL    1     16      6     10      3      9     13     1
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/
```

- (01) LMS is invoked.
- (02) Processing operands LOG, COM and SUM are specified via PAR:
  - a) All messages and statements are to be logged.
  - b) The comparison log is to be output in its entirety and correction statements are to be generated.
  - c) The comparison statistics are to be stored.
- (03) System file SYSOPT, to which LMS outputs the correction statements, is reassigned to file COR.ELEM.
- (04) Program library UEB.BIB is assigned as the I/O library.
- (05) Members DAT and SDAT are to be compared. Subsequently the comparison log is output.
- (06) The comparison statistics (sum field S1) are output.
- (07) System file SYSOPT is reassigned.
- (08) File COR.ELEM, which contains the correction statements, is added as an identically named member of type J.
- (09) Member COR.ELEM is listed.
- (10) LMS is terminated.  
Since SUM was issued during the LMS run (06), sum field S2 is output.

## Output a member to a file

EDT is used to create a SAM file. This file is added as a member and output in the form of three different types of file:

- as a SAM file, on the basis of the file attributes stored in the member;
- as an ISAM file using ISAM keys created by default by setting processing operand FCBTYP=ISAM prior to output;
- as an ISAM file with ISAM keys which are set using the VALUE processing operand. Processing operand FCBTYP must in this case have the value ISAM, as above.

```

/START-PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX!LIB UEB.BIB,BOTH ----- (02)
PAR LOG=MAX
LIB UEB.BIB,BOTH
$EDT ----- (03)
EDT

```

1.00	BACH	SEBASTIAN	MUENCHEN	AUF DER HOEHE 7	AB 3
2.00	BERGMANN	NORBERT	MUENCHEN	TORWEG 10	AB 5
3.00	FINK	SUSANNE	NUERNBERG	RINGSTR. 23	AB 1
4.00	MEYER	FRANZ	NUERNBERG	WASSERMUNGENWEG	AB 1
5.00	GRUNDLER	WOLFGANG	BASEL	SONNENSTR. 11	AB 2
6.00	KNOLL	MONIKA	FRANKFURT	BAUMALLEE 12	AB 3
7.00	LIEDL	ERIKA	MUENCHEN	IN DER BREITE 1	AB 5
8.00	WAGNER	JOHANN	AUGSBURG	AM SEE 45	AB 4
9.00					
10.00					
11.00					
12.00					
13.00					
14.00					
15.00					
16.00					
17.00					
18.00					
19.00					
20.00					
21.00					
22.00					
23.00					
write'pers.dat';halt					0001.00:001(0)

## Examples

---

```
$PAR KEY=YES ----- (04)
PAR KEY=YES
$ADD PERS.DAT>PERDAT ----- (05)
ADD PERS.DAT>PERDAT
INPUT FILE
OUTPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
      ADD PERS.DAT AS (D)PERDAT/@(0001)/1991-08-05

$TOCD * ----- (06)
TOCD *
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
TYP NAME VER (VAR#) DATE
(D) PERDAT @ (0001) 1991-08-05
      1 (D)-ELEMENT(S) IN THIS TABLE OF CONTENTS

$SELD PERDAT ----- (07)
SELD PERDAT
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT FILE
      SEL (D)PERDAT/@(0001)/1991-08-05 AS PERDAT

$SYS SHOW-FILE-ATTRIBUTES PERDAT,INFORMATION=ALL ----- (08)
SYS SHOW-FILE-ATTRIBUTES PERDAT,INFORMATION=ALL
00000003 :N:$USER.PERDAT
FCBTYPE = SAM          VSNTYPE = PUB
LASTPG = 00000001     2ND ALLO= 00003
SHARE = NO            ACCESS = WRITE
ACL = NO              AUDIT = NONE          DESTROY = NO
CRDATE = 1991-08-05  EXDATE = 1991-08-05  LADATE = 1991-08-05
RDPASS = NONE         WRPASS = NONE         EXPASS = NONE
ACCESS# = 001         VERSION = 001
LARGE = NO            BACKUP = A            MIGRATE = ALLOWED
BLKTYPE = STD         BLKSIZE = 002048      BLKCTRL = PAMKEY
RECFORM = (V,N)       RECSIZE = 000000
VSN/DEV/EXT = PUBN02 / D3480 / 001
EXTCNT = 1
:N: PUBLIC: 1 FILE RES= 3 FREE= 2 REL= 0 PAGES
$PAR FCBTYPE=ISAM!SELD PERDAT>PERSDAT ----- (09)
PAR FCBTYPE=ISAM
SELD PERDAT>PERSDAT
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT FILE
      SEL (D)PERDAT/@(0001)/1991-08-05 AS PERSDAT

$SYS SHOW-FILE-ATTRIBUTES PERSDAT,INFORMATION=ALL ----- (10)
SYS SHOW-FILE-ATTRIBUTES PERSDAT,INFORMATION=ALL
00000003 :N:$USER.PERSDAT
FCBTYPE = ISAM          VSNTYPE = PUB
LASTPG = 00000002     2ND ALLO= 00003
SHARE = NO            ACCESS = WRITE
ACL = NO              AUDIT = NONE          DESTROY = NO
CRDATE = 1991-08-05  EXDATE = 1991-08-05  LADATE = 1991-08-05
RDPASS = NONE         WRPASS = NONE         EXPASS = NONE
ACCESS# = 001         VERSION = 001
LARGE = NO            BACKUP = A            MIGRATE = ALLOWED
BLKTYPE = STD         BLKSIZE = 002048      BLKCTRL = PAMKEY
RECFORM = (V,N)       RECSIZE = 000000
KEYLEN = 008          KEYPOS = 00005
VSN/DEV/EXT = PUBN01 / D3480 / 001
EXTCNT = 1
:N: PUBLIC: 1 FILE RES= 3 FREE= 1 REL= 0 PAGES
```

**\$EDT** \_\_\_\_\_ (11)  
EDT

0.10	BACH	SEBASTIAN	MUENCHEN	AUF DER HOEHE 7	AB 3
0.20	BERGMANN	NORBERT	MUENCHEN	TORWEG 10	AB 5
0.30	FINK	SUSANNE	NUERNBERG	RINGSTR. 23	AB 1
0.40	MEYER	FRANZ	NUERNBERG	WASSERMUNGENWEG	AB 1
0.50	GRUNDLER	WOLFGANG	BASEL	SONNENSTR. 11	AB 2
0.60	KNOLL	MONIKA	FRANKFURT	BAUMALLEE 12	AB 3
0.70	LIEDL	ERIKA	MUENCHEN	IN DER BREITE 1	AB 5
0.80	WAGNER	JOHANN	AUGSBURG	AM SEE 45	AB 4
1.80					
2.80					
3.80					
4.80					
5.80					
6.80					
7.80					
8.80					
9.80					
10.80					
11.80					
12.80					
13.80					
14.80					
15.80					
<b>get 'persdat' noreseq</b>					0001.00:001(0)
<b>halt</b>					

**\$PAR VALUE=10000000/200** \_\_\_\_\_ (12)

PAR VALUE=10000000/200

**\$SELD PERDAT>PERKEY3** \_\_\_\_\_ (13)

SELD PERDAT>PERKEY3

INPUT LIBRARY= :N:\$USER.UEB.BIB,DEV=DISK

OUTPUT FILE

SEL (D)PERDAT/@(0001)/1991-08-05 AS PERKEY3

## Examples

---

**\$EDT** \_\_\_\_\_ (14)  
EDT

```
1000.00 BACH          SEBASTIAN    MUENCHEN    AUF DER HOEHE 7    AB 3
1000.02 BERGMANN     NORBERT     MUENCHEN    TORWEG 10          AB 5
1000.04 FINK         SUSANNE     NUERNBERG   RINGSTR. 23        AB 1
1000.06 MEYER        FRANZ       NUERNBERG   WASSERMUNGENWEG   AB 1
1000.08 GRUNDLER     WOLFGANG    BASEL       SONNENSTR. 11      AB 2
1000.10 KNOLL        MONIKA      FRANKFURT   BAUMALLEE 12       AB 3
1000.12 LIEDL        ERIKA       MUENCHEN    IN DER BREITE 1    AB 5
1000.14 WAGNER        JOHANN      AUGSBURG    AM SEE 45          AB 4
1001.14
1002.14
1003.14
1004.14
1005.14
1006.14
1007.14
1008.14
1009.14
1010.14
1011.14
1012.14
1013.14
1014.14
1015.14
get 'perkey3' noreseq          0001.00:001(0)
halt
```

**\$END** \_\_\_\_\_ (15)  
END  
% LMS0311 LMS V02.0A10 ENDED NORMALLY  
/

- (01) LMS is invoked.
- (02) Two statements separated by an exclamation mark are entered:
  - a) All messages and statements are logged.
  - b) Program library UEB.BIB is assigned as the I/O library.
- (03) This format of EDT causes LMS to branch to the editor EDT in order to generate or process a file.

Subsequently the data is entered and stored as SAM file PERS.DAT by means of WRITE. HALT terminates the editor EDT and returns control to LMS.
- (04) Processing operand PAR KEY is set to YES so that all file attributes can be transferred.

- (05) File PERS.DAT is added to the library as a D-type member having the name PERDAT.
- (06) The directory of library UEB.BIB for member type D is to be listed.
- (07) Member PERDAT is output as file PERDAT. Since no file attributes have been specified for this file, LMS generates a SAM file in accordance with the file attributes stored.
- (08) The file attributes of the generated file are listed.
- (09) a) Processing operand FCBTYPE=ISAM defines that the text-based members are to be output as ISAM files.  
b) Member PERDAT is output as ISAM file PERSDAT.
- (10) The file attributes of the generated file are listed.
- (11) LMS branches to EDT. When the ISAM file is listed using EDT, the first six digits of the ISAM key are visible in the line number display.  
  
LMS generated 8-digit ISAM keys with an initial value of 1000 and an increment of 1000.
- (12) The initial value and increment for the ISAM key are specified so that member PERDAT can be output again.
- (13) Member PERDAT is output as ISAM file PERKEY3. Processing operand VALUE is interpreted when the ISAM key is generated because the FCBTYPE=ISAM processing operand in this example is still set.
- (14) LMS branches to EDT. The first six digits of the ISAM key are shown in the line number display in the EDT listing of the ISAM file.
- (15) LMS is terminated.

Examples

Output comparison statistics

All members of several libraries are compared, first with the aid of the traditional cross comparison and subsequently with the Heckel algorithm. In either case, only comparison statistics are output.

```

/SET-FILE-LINK LINK-NAME=LIB001,FILE-NAME=LIB.SOU.V1 }
/SET-FILE-LINK LINK-NAME=LIB002,FILE-NAME=LIB.MAC.V1 } ----- (01)
/SET-FILE-LINK LINK-NAME=LIB003,FILE-NAME=LIB.ALL.V2 }
/START-PROGRAM $LMS ----- (02)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX,COM=/L2/SUM,SUM=YES ----- (03)
PAR LOG=MAX,COM=/L2/SUM,SUM=YES
$COMS *(3)=*(1) ----- (04)
COMS *(3)=*(1)

-----
PRIMARY ELEMENT= (S)EINAUS/@(0001)/1991-08-09
SECONDARY ELEMENT= (S)EINAUS/@(0003)/1991-08-09
RESULT: S PRIMARY= 8 INSERTED= - ( -) DELETED= - ( -)
SECONDARY= 8 SAME= 8 ( 1)
-----
PRIMARY ELEMENT= (S)ERFASS/@(0002)/1991-08-09
SECONDARY ELEMENT= (S)ERFASS/@(0002)/1991-08-09
RESULT: S PRIMARY= 14 INSERTED= - ( -) DELETED= - ( -)
SECONDARY= 14 SAME= 14 ( 1)
-----
PRIMARY ELEMENT= (S)PROT/@(0001)/1991-08-09
SECONDARY ELEMENT= (S)PROT/@(0003)/1991-08-09
RESULT: S PRIMARY= 5 INSERTED= - ( -) DELETED= - ( -)
SECONDARY= 5 SAME= 5 ( 1)
-----
} ----- (05)
} ----- (06)
$SUM ----- (06)
SUM
AREA C1
-----
STATISTIC PRIM. PRIM. INS. SAME DEL. INS+DEL SEC. SEC.
ELEM. LINES LINES LINES LINES LINES LINES ELEM.
S (SAME) 3 27 - 27 - - 27 3
C (CHANGED) 0 0 0 0 0 0 0 0
I (INSERTED) 0 0 0 - - 0 - -
D (DELETED) - - - - 0 0 0 0
-----
TOTAL 3 27 0 27 0 0 27 3
$PAR COM= ----- (07)
PAR COM=
$COMM *(3)=*(2) ----- (08)
COMM *(3)=*(2)
FUNCTION = C O M P A R E
PAR COMPARE= 00001/00072/L/MED
PRIMARY LIBRARY= :N:$USER.LIB.ALL.V2
PRIMARY ELEMENT= (M)MAC1/@(0002)/1991-08-09
SECONDARY LIBRARY= :N:$USER.LIB.MAC.V1
SECONDARY ELEMENT= (M)MAC1/@(0003)/1991-08-09

```



```

SAME FROM          #1 TO          #5 AS FROM          #1 TO          #5
-----
PRIMARY  ELEMENT= (M)MAC1/@(0002)/1991-08-09
SECONDARY ELEMENT= (M)MAC1/@(0003)/1991-08-09
RESULT: S   PRIMARY= 5 INSERTED= - ( -) DELETED= - ( -)
            SECONDARY= 5 SAME= 5 ( 1)
FUNCTION    = C O M P A R E
PAR         COMPARE= 00001/00072/L/MED
PRIMARY    LIBRARY= :N:$USER.LIB.ALL.V2
PRIMARY    ELEMENT= (M)MAC2/@(0001)/1991-08-09
SECONDARY  LIBRARY= :N:$USER.LIB.MAC.V1
SECONDARY  ELEMENT= (M)MAC2/@(0002)/1991-08-09
    
```

} - (09)

```

SAME FROM          #1 TO          #5 AS FROM          #1 TO          #5
-----
PRIMARY  ELEMENT= (M)MAC2/@(0001)/1991-08-09
SECONDARY ELEMENT= (M)MAC2/@(0002)/1991-08-09
RESULT: S   PRIMARY= 5 INSERTED= - ( -) DELETED= - ( -)
            SECONDARY= 5 SAME= 5 ( 1)
FUNCTION    = C O M P A R E
PAR         COMPARE= 00001/00072/L/MED
PRIMARY    LIBRARY= :N:$USER.LIB.ALL.V2
PRIMARY    ELEMENT= (M)MUC1/@(0001)/1991-08-09
    
```

} - (09)

```

PRIMARY  ELEMENT= (M)MUC1/@(0001)/1991-08-09
RESULT: I   PRIMARY= 5 INSERTED= 5 ( 1) DELETED= - ( -)
            SECONDARY= - SAME= - ( -)
FUNCTION    = C O M P A R E
PAR         COMPARE= 00001/00072/L/MED
PRIMARY    LIBRARY= :N:$USER.LIB.ALL.V2
PRIMARY    ELEMENT= (M)MUC2/@(0001)/1991-08-09
    
```

} - (09)

```

PRIMARY  ELEMENT= (M)MUC2/@(0001)/1991-08-09
RESULT: I   PRIMARY= 5 INSERTED= 5 ( 1) DELETED= - ( -)
            SECONDARY= - SAME= - ( -)
    
```

} - (09)

\$SUM ----- (10)

SUM

AREA C1

STATISTIC	PRIM. ELEM.	PRIM. LINES	INS. LINES	SAME LINES	DEL. LINES	INS+DEL LINES	SEC. LINES	SEC. ELEM.
S (SAME)	2	10	-	10	-	-	10	2
C (CHANGED)	0	0	0	0	0	0	0	0
I (INSERTED)	2	10	10	-	-	10	-	-
D (DELETED)	-	-	-	-	0	0	0	0
TOTAL	4	20	10	10	0	10	10	2

\$END ----- (11)

END

AREA C2

STATISTIC	PRIM. ELEM.	PRIM. LINES	INS. LINES	SAME LINES	DEL. LINES	INS+DEL LINES	SEC. LINES	SEC. ELEM.
S (SAME)	5	37	-	37	-	-	37	5
C (CHANGED)	0	0	0	0	0	0	0	0
I (INSERTED)	2	10	10	-	-	10	-	-
D (DELETED)	-	-	-	-	0	0	0	0
TOTAL	7	47	10	37	0	10	37	5

## Examples

---

```
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/
```

- (01) The libraries to be compared are assigned.
- (02) LMS is invoked.
- (03) Processing operands LOG, COM and SUM are specified via PAR:
  - a) All messages and statements are logged.
  - b) The cross comparison method is branched into. No comparison log is created; only the results are output.
  - c) The comparison statistics are stored.
- (04) All type-S members of program library LIB.ALL.V2 are compared with the members of source library LIB.SOU.V1.
- (05) The result of the comparison is output.
- (06) SUM is used to output sum field S1, which contains the comparison statistics of the entire comparison. The comparison statistics show, for example, that a total of 56 records have been compared.
- (07) The Heckel algorithm is returned to (default value).
- (08) All type-M members of program library LIB.ALL.V2 are compared with the members of macro library LIB.MAC.V1.
- (09) The result of the comparison is output.
- (10) The comparison statistics of the entire comparison are output.
- (11) LMS is terminated.

Sum field S2 is output automatically. The two sets of comparison statistics are added in sum field S2. Hence these statistics show the results of all comparisons made in the LMS run.

### Branch to a user program while a member is being listed

The user program lists only the first 10 input records of a member.  
If a member consists of less than 10 records, the program fills it with additional records to bring it up to 10.

```

/START-PROGRAM $LMS ----- (01)
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$PAR LOG=MAX!LIB TEST.LIB,BOTH ----- (02)
PAR LOG=MAX
LIB TEST.LIB,BOTH
$LSTS USELST ----- (03)
LSTS USELST
INPUT LIBRARY= :N:$USER.TEST.LIB,DEV=DISK
INPUT ELEMENT= (S)USELST@(0001)/1991-08-09
      TITLE 'USER EXIT FOR THE FUNCTION: LST'
*
*
*       1.) THIS SUBROUTINE HAS THE EFFECT THAT ONLY THE FIRST 10
*           RECORDS OF EACH MEMBER ARE LISTED.
*
*       2.) IF THE MEMBER CONTAINS LESS THAN 10 RECORDS,
*           ADDITIONAL RECORDS ARE INSERTED.
* INPUT FROM LMS: R1=A(PARAMETER LIST)
*                 R13=A(SAVE AREA), 18 WORDS
*                 R14=RETURN ADDRESS
*                 R15=A(USER PROGRAM)
*
PARDSEC  DSECT
AUFTRAG  DS    A                A(JOB FROM LMS)
*
*
*
*
ANTWORT  DS    A                A(RESPONSE FROM USER PROGRAM)
*
*
*
*
SATZ     DS    A                A(RECORD, INCL. 4-BYTE HEADER)
*
*
PARDSECL EQU  *-PARDSEC        L'DSECT
USELST   CSECT PAGE
          STM  0,15,0(13)       SAVE REGISTERS
          LR   10,15            BASE
          USING PARDSEC,1       LMS PARAMETER LIST
          USING USELST,10
          L    6,AUFTRAG        A(JOB)
          L    7,ANTWORT        A(RESPONSE)
          L    8,SATZ           A(RECORD)
          CLC  0(3,6),REC       RECORD OFFERED ?
          BE   DOSATZ           YES ——?
          CLC  0(3,6),BOE       START OF ELEMENT
          BE   DOBOE            YES ——?
          CLC  0(3,6),EOE       END OF ELEMENT
          BE   DOEOE            YES ——?

```

## Examples

```

      B      RETURN
*
DOBOE  EQU   *
      ZAP  ANZAHL,P0      COUNTER := 0
      B      RETURN
*
DOSATZ EQU   *
      CP   ANZAHL,P10     ALREADY 10 RECORDS OUTPUT ?
      BNL  DODEL          YES (SKIP REMAINDER) ?
      AP   ANZAHL,P1     COUNTER := COUNTER +1
      B      DOCON
*
DOEOE  EQU   *
      CP   ANZAHL,P10     ALREADY 10 RECORDS OUTPUT ?
      BNL  DOCON          YES (NO INSERTION) ?
      AP   ANZAHL,P1     COUNTER := COUNTER +1
      B      DOINS
*
DOINS  EQU   *
      MVC  0(3,7),INS     INSERT RECORD
      LA   9,INSSATZ
      ST   9,SATZ         A(RECORD TO BE INSERTED)
      B      RETURN
*
DODEL  EQU   *
      MVC  0(3,7),DEL     DELETE RECORD
      B      RETURN
*
DOCON  EQU   *
      MVC  0(3,7),CON     CONTINUE
*
RETURN EQU   *
      LM   0,15,0(13)     RESTORE REGISTERS
      BR   14
      TITLE 'CONSTANTS AND VARIABLES'
BOE    DC   'BOE'         START OF ELEMENT
REC    DC   'REC'         RECORD OFFERED
EOE    DC   'EOE'         END OF ELEMENT
CON    DC   'CON'         CONTINUE
DEL    DC   'DEL'         DELETE RECORD
INS    DC   'INS'         INSERT NEW RECORD
ANZAHL DC   PL2'0'
P0     DC   PL2'0'
P1     DC   PL2'1'
P10    DC   PL2'10'
INSSATZ DC  Y(INSSATZE-INSSATZ)
      DC   XL2'4040'
      DC   '***** INSERT BY USER-PROGRAM *****'
INSSATZE EQU *
      LTORG
      END
NUMBER OF PROCESSED RECORDS IS      89
$USE LST=TEST.LIB(USELST) (04)
USE LST=TEST.LIB(USELST)
$LSTS EINAUS (05)
LSTS EINAUS
INPUT  LIBRARY= :N:$USER.TEST.LIB,DEV=DISK
INPUT  ELEMENT= (S)EINAUS/@(0001)/1991-08-09

```

```

USER EXIT TEST.LIB(USELST) FOR LSTE IS ACTIVE
      TITLE 'DATA ENTRY'
      PRINT NOGEN
ERFAS  START
      BALR 5,0
      USING *,5
      OPEN DATEI,OUTPUT
LESEN  RDATA SATZ,ENDPGM
      CLC TEXT(4),=C'/EOF'
      BE  ENDPGM
      MVC ATEXT,TEXT
NUMBER OF PROCESSED RECORDS IS      10
$LSTS PERSDAT (06)
LSTS PERSDAT
INPUT  LIBRARY= :N:$USER.TEST.LIB,DEV=DISK
INPUT  ELEMENT= (S)PERSDAT/@(0001)/1991-08-09
USER EXIT TEST.LIB(USELST) FOR LSTE IS ACTIVE
BACH   SEBASTIAN      MUENCHEN      AUF DER HOEHE 7      AB 3
BERGMANN NORBERT      MUENCHEN      TORWEG 10      AB 5
FINK   SUSANNE       NUERNBERG     RINGSTR. 23      AB 1
MEYER  FRANZ         NUERNBERG     WASSERMUNGENWEG  AB 1
GRUNDLER WOLFGANG     BASEL        SONNENSTR. 11    AB 2
KNOLL  MONIKA        FRANKFURT     BAUMALLEE 12    AB 3
LIEDL  ERIKA         MUENCHEN      IN DER BREITE 1  AB 5
WAGNER JOHANN        AUGSBURG      AM SEE 45       AB 4
***** INSERT BY USER-PROGRAM *****
***** INSERT BY USER-PROGRAM *****
NUMBER OF PROCESSED RECORDS IS      10
$END (07)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
/

```

- (01) LMS is invoked.
- (02) Two statements separated by an exclamation mark are entered:
  - a) All messages and statements are logged.
  - b) Program library TEST.LIB is assigned as the I/O library.
- (03) User source program USELST is listed.
- (04) Before listing an input record LMS branches to user program USELST, which resides in library TEST.LIB.
- (05) The first 10 records of member EINAUS of the assigned program library TEST.LIB are listed.
- (06) Member PERSDAT is listed. Since it is shorter than 10 records it is padded with records by the user program.
- (07) LMS is terminated.



---

## Old LMS subroutine interface

The old subroutine interface described here continues to be supported for reasons of compatibility.

The new subroutine interface is described in the manual "LMS Subroutine Interface" [15]

If LMS is invoked as a subroutine with a separate prompting facility, it returns control to the calling program after END has been processed. LMS remains loaded after END has been processed. Apart from that, execution is the same as when LMS is loaded with the /START-PROGRAM command.

Whenever LMS is invoked, it starts its own STXIT routine. On return to the main program the LMS STXIT routine is terminated; the main program must reactivate its own STXIT routine.

The following register conventions must be observed when calling the subroutine interface:

Register 1	must be zero.
Register 13	contains the address of a save area (comprising 18 words) which must be provided by the calling program. This area is used by LMS to store the registers of the calling program.
Register 14	contains the return address.
Register 15	contains the entry address LMSUP.

Before returning control to the calling program, LMS places the following return codes in register 15:

X'00'	LMS terminated normally (corresponds to TERM termination).
X'04'	LMS terminated abnormally (corresponds to TERMJ termination; cf. PAR TERMINATE, page 246).

## Old LMS subroutine interface

---

### Example

LMS is invoked as a subroutine from the program UPROG. In LMS, a member is transferred from a program library to a file. After termination of the LMS run, control is returned to the user program.

```
/START-PROGRAM $LMS
% BLS0500 PROGRAM 'LMS', VERSION 'V02.0A10' OF '91-05-29' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990.
% ALL RIGHTS RESERVED
% LMS0310 LMS VERSION V02.0A10 LOADED
$LIB UEB.BIB,BOTH!LSTS LMSCALL _____ (01)
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
INPUT ELEMENT= (S)LMSCALL/@(0001)/1991-08-12
UPROG START
      BALR 3,0
      USING *,3
      MVC OUTPUT,ANMELD
AUFBRUF WROUT OUT,TERM
      LA 14,RUECK _____ (02)
      LA 1,0 _____ (03)
      LA 13,SAVE _____ (04)
      L 15,=V(LMSUP) _____ (05)
      BALR 14,15
RUECK MVC OUTPUT,ABMELD
      WROUT OUT,TERM
*
*
TERM TERM
*
*
SAVE DS 18F
*
ANMELD DC '***** *LMS IS BEING CALLED *****'
ABMELD DC '***** LMS TERMINATED - PROGRAM CONTINUES *****'
OUT DC Y(ENDE-OUT)
      DS CL2
      DC X'01'
OUTPUT DS CL50
ENDE EQU *
*
*
      END UPROG
NUMBER OF PROCESSED RECORDS IS 29
$END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
.
.
.
/START-PROGRAM FROM-FILE=(LIB=UEB.BIB,ELEM=LMSCALL)
% BLS0500 PROGRAM 'LMSCALL', VERSION '007' OF '91-08-12' LOADED
***** *LMS IS BEING CALLED *****
```



```

% LMS0310 LMS VERSION V02.0A10 LOADED ----- (06)
$PAR LOG=MAX!LIB UEB.BIB,BOTH
PAR LOG=MAX
LIB UEB.BIB,BOTH
$TOCC *
TOCC *
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
TYP NAME VER (VAR#) DATE
(C) LMSCALL 007 (0001) 1991-08-12
1 (C)-ELEMENT(S) IN THIS TABLE OF CONTENTS
$SELS LMSCALL>LMSCALL1
SELS LMSCALL>LMSCALL1
INPUT LIBRARY= :N:$USER.UEB.BIB,DEV=DISK
OUTPUT FILE
SEL (S)LMSCALL/@(0001)/1991-08-12 AS LMSCALL1
$END ----- (07)
END
% LMS0311 LMS V02.0A10 ENDED NORMALLY
***** LMS TERMINATED - PROGRAM CONTINUES *****
/

```

- (01) Two statements separated by an exclamation mark are input:
  - a) Program library UEB.BIB is assigned as the I/O library.
  - b) Source program LMSCALL is listed.
- (02) The return address is loaded into register 14.
- (03) Register 1 is set to 0.
- (04) The address of the save area is loaded into register 13.
- (05) The entry address LMSUP is loaded into register 15.
- (06) LMS is invoked from the user program.
- (07) After termination of the LMS run, control is returned to the user program.



## Messages

The error messages are output in the form of a 7-character code (LMSnnnn) and help texts explaining their meaning and any necessary action.

### List of messages

LMS0001      START: NOT ENOUGH MEMORY FOR LMS

**Meaning**

Contact the system administrator.

LMS0002      LIB ASSIGNMENT LOST DURING LMS RUN

LMS0003      INVALID STATEMENT

**Meaning**

Possible error causes:

- blank between operation and operand missing
- the statement is unknown to LMS.

LMS0004      BLANK MISSING IN FRONT OF OPERAND

LMS0005      MEMBER TYPE INVALID OR NOT ASSIGNED

LMS0006      END OR CTL STATEMENT EXPECTED

LMS0007      ELEMENT CONTAINS FORMAT-B RECORDS

LMS0008      SYNTAX ERROR IN OPERAND

LMS0009      SYNTAX ERROR IN STATEMENT

LMS0010      SYNTAX ERROR IN DATE

**Meaning**

The syntax for date is YYYY-MM-DD or YYMMDD.

LMS0011 SYNTAX ERROR IN VERSION NUMBER

**Meaning**

The syntax required when specifying the version number of the current library type is described in the LMS manual.

LMS0012 SYNTAX ERROR IN NAME

LMS0013 SYNTAX ERROR IN (LIB)

LMS0014 LENGTH OF RECORDTYPE 163 MUST BE BETWEEN 36 AND 44

**Meaning**

Record type 163 consists of 1-32 bytes secondary name followed by 0-8 bytes secondary attribute.

LMS0015 (LIB) WRONG OR NOT ASSIGNED

LMS0016 INPUT LIBRARY NOT ASSIGNED OR SPECIFICATION INVALID

LMS0017 FUNCTION NOT YET IMPLEMENTED

LMS0018 WARNING: USE CORRECT SYNTAX FOR STATEMENT

LMS0019 REQUESTED MEMBER OF TYPE 'J' IN CTL STATEMENT DOES NOT EXIST

LMS0020 REQUESTED MEMBER DOES NOT EXIST IN SPECIFIED LIBRARY

LMS0021 LIBRARY IS DESTROYED

**Response**

Try to duplicate the members into a new library using the DUP statement.

LMS0022 NOT ENOUGH SPACE IN LIBRARY

**Meaning**

Limits of the library have been reached.

**Response**

Reorganize the library.

LMS0023 DMS ERROR CODE '(&00)'. ERROR INFO IN SYSTEM MODE: /HELP DMS(&00),INF=D

**Meaning**

Repeated attempts proved unsuccessful.

For more detailed information on the DMS error enter the /HELP command in system mode or see the BS2000 manual 'System Messages', 'DMS Disk Processing' or 'DMS Tape Processing'.

**Response**

See the DMS error code.

LMS0024 ACCESS ERROR ON '(&00)' LIBRARY \*\*\* (&01)

**Meaning**

AMCB0010: Address outside of member ; AMCB0016: Compress flag invalid  
 AMCB0017: Last member erased ; AMCB0018: No OSM library  
 AMCB0025: No DIR2 entry created ; AMCB0027: No new FP chain created  
 AMCB0052: Element replaced ; AMCB0054: Empty file replaced  
 AMCB0108/AMCB109: USER/OPEN-error ; AMCB0120: Filename invalid  
 AMCB0121: No FT entry for file ; AMCB0122: Open state conflict  
 AMCB0125: Second access on out. lib.; AMCB0134: CTL element already opened  
 AMCB0137: Second access on seq. library  
 AMCB0131: Library is already opened for CTL or PRT  
 AMCB0255: No SVC PLAM available

LMS0026 FILE IS NOT A LIBRARY

**Meaning**

The file specified in the LIB statement is not a library.

LMS0027 MEMBER TYPE NOT PERMITTED FOR '(&00)' LIBRARY

LMS0029 ORDER OF MEMBER TYPES FOR SEQUENTIAL LIBRARY INVALID. STATEMENT NOT PROCESSED

**Meaning**

When writing to a sequential library the order R, M, S for member types has not been observed.

LMS0030 VSN OF VOLUME DOES NOT AGREE WITH SPECIFICATION IN LIBIN OR LIBOUT STATEMENT

**Response**

Check the VSN.

LMS0031 INTERNAL ERROR. AMCB ERROR CODE '(&00)'

**Meaning**

AMCB0002: Invalid OP code  
 AMCB0003: Missing filename in control block  
 AMCB0004: No/changed FCB address  
 AMCB0007/AMCB0012: Conflicting data in control block/FCB  
 AMCB0013/AMCB0014: Supplementary information missing/invalid  
 AMCB108/AMCB0109: User/OPEN error  
 AMCB0127: LT entry missing  
 AMCB0136: Access error e.g. locked file  
 AMCB0141: Unknown access method

LMS0033 INTERNAL ERROR WHEN ASSIGNING '(&00)' FILE

LMS0036 WARNING: LIBRARY TO BE CLOSED IS NOT ASSIGNED

**Response**

Check the library name.

LMS0038 READ ERROR '(&00)' WHEN READING MEMBER RECORD

**Meaning**

In the case of a system error, enter /HELP in system mode or see the BS2000 manual 'System Messages', 'DMS Disk Processing', or 'DMS Tape Processing'.

LMS0039 INPUT AND OUTPUT LIBRARY MAY NOT BE IDENTICAL FOR SEQUENTIAL LIBRARIES

**Response**

See the COR statement.

LMS0040 WARNING: INVALID VERSION. VERSION '001' ASSUMED

**Meaning**

Error when specifying the version, e.g. the 2nd or 3rd byte is not a digit. Version '001' is used.

LMS0041 NOT ENOUGH MEMORY TO PROCESS STATEMENT

LMS0043 CONTROL NUMBER DOES NOT MATCH THE ONE COMPUTED BY LMS

**Meaning**

The LMS cross control number can be obtained with the aid of TEST-MODE.

**Response**

Check the UPD statement.

LMS0047 UPDATE NOT (COMPLETELY) IN TEXT AREA

**Response**

Check the UPD statement.

LMS0048 SPECIFIED CORRECTION ADDRESS DOES NOT EXIST

**Meaning**

Likely error cause: the address specified in the UPD statement is too small or too large.

**Response**

Check the UPD statement.

LMS0049 CONTROL TEXT NOT AT SPECIFIED ADDRESS

**Response**

Check the TXT records using the LST statement and correct the UPD statement accordingly.

LMS0050 ASSIGNMENT FOR (&00) MISSING OR INVALID

LMS0051 OUTPUT LIBRARY WRONG OR NOT ASSIGNED

**Response**

Assign an output library using LIB...,OUT and reenter the statement.

LMS0052 SYNTAX ERROR IN SPECIFIED ABBREVIATION (LIB)

LMS0054 (LIB) WRONG OR NOT ASSIGNED - (&00)

**Meaning**

Library not found in TFT - library not assigned.

Assigned file is no library.

No library existing.

AMCB0102: Invalid file type

AMCB0109: OPEN error

AMCB0150: Unknown access method

**Response**

Assign the library with FILE command or assign a correct one

LMS0055 INPUT MEMBERS SPECIFIED IN COM STATEMENT MAY NOT BE IN THE SAME SEQUENTIAL LIBRARY

LMS0057 CROSS CONTROL NUMBER NOT DEFINED

**Meaning**

Assign the cross control number using \*CON.

LMS0059 MEMBER TYPE NOT PERMITTED FOR THIS FUNCTION

**Response**

Member types permitted for this function are listed in the LMS manual.

LMS0060 INPUT AND OUTPUT PERFORMED FOR THE SAME FILE

**Meaning**

It is not permitted to assign the same file to both input and output library.

LMS0061 INPUT AND OUTPUT MAY NOT BE PERFORMED FOR THE SAME SEQUENTIAL LIBRARY

LMS0062 ILLEGAL STATEMENT FOR SEQUENTIAL INPUT LIBRARY. PROCESSING CONTINUES WITH NEXT STATEMENT

**Meaning**

For instance, a NAM or DEL statement is not allowed.

LMS0063 LOG OUTPUT TO SEQUENTIAL INPUT/OUTPUT LIBRARY INVALID

**Meaning**

Log output to a sequential library serving as input/output library is not allowed.

LMS0065 SYNTAX ERROR IN SPECIFIED VSN

**Meaning**

Possible error causes:

- (1) no brackets have been specified
- (2) the specification does not consist of 6 characters
- (3) (vsn) is not the last operand in the LIBOUT statement.

LMS0066 WARNING: AT LEAST ONE NON-NUMERIC ISAM KEY GENERATED

**Meaning**

In the EDTx statement at least one record has been output with a non-numeric ISAM key that cannot be processed by EDT.

**Response**

Either use the COR statement or delete such records.

LMS0067 TYPE 'R' NOT ALLOWED FOR OSM LIBRARIES

LMS0068 FUNCTION NOT ALLOWED FOR THIS LIBRARY TYPE

**Meaning**

For instance, the PRT or CTL statement is not allowed for tape or OML libraries.

LMS0072 WARNING: NO NAME TO BE PROCESSED

LMS0075 WARNING: NO RANGE FIELD DEFINED FOR OUTPUT RECORDS. NUMBERING NOT PERFORMED

**Meaning**

Although VALUE has been specified, no range field has been defined for the output records.

**Response**

Supply the RANGE operand with values.



LMS0077 WARNING: STRING LONGER THAN CHECK FIELD. NO STRING INSERTED.

**Meaning**

The length specified for the check field of the output records is shorter than the string. Processing continues, but the string is ignored.

**Response**

Adjust the values of the RANGE and STRING operands.

LMS0078 WARNING: OVERFLOW WHILE RENUMBERING

**Meaning**

Renumbering has been performed; however, a number overflow occurred.

LMS0079 WARNING: CHECK FIELDS IN INPUT RECORDS NOT ASCENDING

LMS0086 NAME OR OPERAND IS TOO LONG

**Meaning**

The specified name or operand may only have up to 8 characters.

LMS0090 INPUT ALLOWED ONLY FROM IMPLICIT INPUT LIBRARY

**Meaning**

An explicit abbreviation is prohibited.

LMS0091 WARNING: NO RENUMBERING PERFORMED. NUMERIC VALUE FOR VALUE OPERAND TOO HIGH

**Response**

Compare with the specified RANGE.

LMS0093 MEMBER FOR LOG OUTPUT ALREADY EXISTS

**Meaning**

Log output (PRT statement) to a member that already exists is not possible if OVERWRITE=NO is set.

LMS0095 INPUT DATA RECORDS ARE MISSING

LMS0097 VALUE OR STRING OPERAND MISSING. STATEMENT NOT PROCESSED

**Response**

Supply the appropriate operands with a value.

LMS0098 INCONSISTENT OPERANDS. STATEMENT NOT PROCESSED

LMS0099 OPERAND RANGE=NO SPECIFIED. STATEMENT NOT PROCESSED

LMS0100 INVALID DELIMITER

LMS0101 AT LEAST ONE OPERAND MISSING IN STATEMENT

LMS0102 WARNING: AT LEAST ONE INCOMPLETE MODULE FOUND IN \*OMF

**Meaning**

When transferring modules from the EAM object module library using the 'ADDR \*OMF' statement at least one incomplete module has been detected (e.g. due to an aborted assembler run).

LMS0103 RECORD(S) OF ILLEGAL TYPE WERE REMOVED

LMS0104 NAME IS TOO LONG

LMS0105 COLUMN IN '\*CHANGE...' OF COR STATEMENT INCOMPLETE

LMS0106 COLUMN IN '\*CHANGE...' OF COR STATEMENT TOO LONG

LMS0107 LENGTH OF CONTROL TEXT NOT EQUAL TO LENGTH OF UPDATE TEXT

**Meaning**

If '=:' has been specified when replacing, the control text and the update text must have the same length.

LMS0108 INVALID REPLY FROM USER PROGRAM. CORRECT PROGRAM

**Meaning**

The reply given by a user program connected using USE is not CON, DEL or INS.

LMS0109 WARNING: LIST MEMBER IN OLD FORMAT DESTROYED

**Meaning**

If desired, copy the list element and assign it again, or repeat the operation.

LMS0110 UPDATE TEXT MISSING

LMS0111 MODIFICATION MISSING

LMS0113 CONTROL TEXT IS PARTLY OR ENTIRELY IN RANGE FIELD

LMS0114 UPDATE TEXT IS PARTLY OR ENTIRELY IN RANGE FIELD

LMS0116 ENTRY-NAME OF USER EXIT MAY NOT START WITH 'LMS'

**Response**

Correct the entry name.

LMS0117 COLUMN < 1 OR > 80 INVALID

LMS0118 LMS TERMINATED ABNORMALLY

LMS0119 FIRST COLUMN CANNOT BE HIGHER THAN SECOND COLUMN

LMS0121 RECORD NUMBER IN CORRECTION STATEMENT INCORRECT

**Meaning**

The record number has this format: #<digit>.  
<digit>: max. positive integer of 8 bytes.

LMS0123 COLUMN IS IN RANGE FIELD

LMS0124 NO COLUMN SPECIFIED

LMS0125 RIGHT STRING DELIMITER MISSING

LMS0126 ILLEGAL /INTR COMMAND SPECIFIED

LMS0127 LENGTH OF CHECK FIELD DOES NOT MATCH CHECK LENGTH

LMS0128 NO CHECK FIELD DEFINED. SUBSEQUENT RECORD INSERTED AT CURRENT POSITION

LMS0129 WARNING: LMS STATEMENT ABORTED DUE TO INTERRUPT

LMS0131 MEMBER EMPTY AFTER PROCESSING COR STATEMENT. MEMBER NOT CORRECTED

**Meaning**

While processing a COR statement, all the records of the member have been deleted.

LMS0132 PROCESSED MEMBER STILL IN FILE '(&00)'. USE ADD STATEMENT TO ADD FILE TO LIBRARY

**Meaning**

It is not possible to rewrite the member upon returning from the editor.

LMS0133 LMS CANNOT CALL EDT. REDEFINE '@' ESCAPE CHARACTER

LMS0134 EDITOR TERMINATED ABNORMALLY

**Meaning**

Element records with record length = 256 existing.  
Element records with record length = 0 existing.

LMS0135 (LIB) OF PRIMARY MEMBER WRONG OR NOT ASSIGNED

LMS0138 CHARACTER '(&00)' MISSING IN OPERAND

LMS0139 LIBIN ASSIGNMENT WRONG OR MISSING

LMS0140 WARNING: SPECIFIED STRING LONGER THAN ISAM KEY. NO STRING INSERTED

LMS0141 CRT STATEMENT NO LONGER SUPPORTED. USE COR STATEMENT

LMS0142 WARNING: CHARACTER(S) NOT EQUAL TO 'BLANK' LOST IN FOLLOWING RECORD

**Meaning**

In the following record characters that were not blanks were lost when text was inserted or replaced.

LMS0143 WARNING: ISAM KEY TRUNCATED FROM RIGHT WHEN MOVED TO CHECK FIELD

LMS0144 KEY NOT ASCENDING OR OVERFLOW DURING NUMBERING

**Meaning**

The content of the check field used to establish the ISAM key is not ascending, or an overflow occurred when generating the ISAM key using the VALUE operand.

LMS0145 MEMBER NAME NOT PERMITTED FOR PLAM LIBRARY

**Meaning**

E.g. the member name for PLAM libraries differs from that for OSM libraries.

LMS0146 LOG OUTPUT NOT POSSIBLE, AS OVERWRITE=ONLY SPECIFIED THOUGH NO MEMBER WITH SAME NAME EXISTS

**Meaning**

Log output (PRT or OUT statement) is not possible in conjunction with OVERWRITE=ONLY, if no member exists with the same name.

LMS0147 TEXT AREA TO BE CORRECTED OVERLAPS REP AREA SPECIFIED IN REP STATEMENT

LMS0148 NEWLIB OPERAND MISSING IN LIBOUT STATEMENT FOR SEQUENTIAL LIBRARY

**Meaning**

In the case of output to sequential libraries the NEWLIB operand must always be specified in the LIBOUT statement.

LMS0149 EDT OR EDOR CANNOT BE LOADED

LMS0150 USER ROUTINE CANNOT BE LOADED

**Meaning**

The user routine does not exist.

LMS0151 WARNING: DEFAULT VALUES GENERATED FOR INCORRECT CTL OR PRT STATEMENT

LMS0152 USER EXIT FOR COM STATEMENT NOT ALLOWED WHEN GENERATING COR STATEMENTS FROM COM

LMS0153 GENERATION OF 'COR' STATEMENT ABORTED PREMATURELY

LMS0154 WARNING: A RECORD NUMBER WAS GENERATED IN AT LEAST ONE COR STATEMENT

**Meaning**

Despite PAR CHECK not equal NO a record number was generated.

LMS0155 NO CORRECTION BY COR SUBSTATEMENT POSSIBLE

**Meaning**

Identification or record number is lower than current identification or record number.

LMS0156 CHANGE CONTINUATION STATEMENT EXPECTED

LMS0157 MEMBER TYPE ONLY ALLOWED FOR PLAM-LIBRARIES

LMS0158 SYS STATEMENT CANNOT BE PROCESSED BECAUSE OF INCORRECT '(&00)'

LMS0159 FMS OR \$FMSLIB DOES NOT EXIST

**Meaning**

Either the FMS does not exist in the module library used by the DLL or no \$FMSLIB exists.

LMS0160 THE OUTPUT OF THE ELEMENT AND THE LMS LOG ARE NOT PERMITTED TO SAME OSM LIBRARY

LMS0162 RANGE LIMITS IN LIBRARY WILL SOON BE REACHED (SATURATION). REORGANIZE LIBRARY

LMS0163 WARNING: AT LEAST ONE RECORD TRUNCATED

LMS0164 WARNING: TABLE OF CONTENTS (TOC) COULD NOT BE UPDATED DUE TO ABNORMAL PROGRAM TERMINATION

**Response**

Open the library again in write mode in order to automatically update the table of contents (TOC).

LMS0165 WARNING: TABLE OF CONTENTS (TOC) HAS BEEN UPDATED

**Meaning**

The library was opened in write mode and the table of contents (TOC) has been updated.

LMS0166 OVERWRITE OPERAND HAS ILLEGAL VALUE

**Meaning**

V, D or EXTENT has been specified.

LMS0167 SPECIFIED LINK NAME CANNOT BE ASSIGNED TO ANY FILE

**Response**

Assign a file to the link name.

LMS0168 FILE FORMAT OF '(&00)' FILE NOT SUPPORTED FOR THIS STATEMENT

**Meaning**

The FCB type of the file to be added ist not allowed.

LMS0169 DMS ERROR '(&00)' ON PROCESSING '(&01)' FILE. ERROR INFO IN SYSTEM MODE:  
/HELP DMS (&00), INF=D

**Meaning**

For more detailed information on the DMS error enter the /HELP command in system mode or see the BS2000 manual 'System Messages', 'DMS Disk Processing' or 'DMS Tape Processing'.

LMS0170 NEXT RECORD IN INPUT MEMBER NOT IN CORRECT ORDER

**Meaning**

The input member is not numbered in ascending order.

LMS0171 RECORD NUMBER OR CHECK FIELD IN PRECEDING CORRECTION STATEMENT NOT IN CORRECT ORDER. CORRECTION NOT PERFORMED

**Meaning**

The correction statements for COR are not sorted in ascending order. The operand in the preceding correction statement for COR is not higher than the operands specified in previous correction statements.

LMS0172 NO CHECK FIELD DEFINED. INPUT MEMBER NOT CORRECTED

**Meaning**

Although CHECK=NO has been set, an identification has been specified in a correction statement. No default values for CHECK are used.

LMS0173 RECORD NUMBER OR CHECK FIELD IN CORRECTION STATEMENT NOT IN CORRECT ORDER

**Meaning**

The record number or the check field in the next record of the correction statement is not higher than those specified for COR in preceding statements.

LMS0174 RECORD NUMBER OR CHECK FIELD IN NEXT DATA RECORD LOWER THAN CURRENT POSITION IN INPUT MEMBER. NO CORRECTION USING 'COR' POSSIBLE

LMS0175 WARNING: EDITOR CALLED, BUT LMS IS IN TEST MODE. NO CORRECTIONS MADE.

**Meaning**

Test mode has been set and the updated element will not be stored into the library.

**Response**

Set TEST=NO or use a correct procedure and edit again.

LMS0176 WARNING: ONLY INPUT MEMBER WITH HIGHEST VERSION IN FMS LIBRARY INCLUDED IN LIBRARY

LMS0177 TOO MANY DELIMITERS SPECIFIED

LMS0178 INTERNAL ERROR '(&00)' WHEN WRITING A RECORD  
LMS0179 INPUT MEMBER '(&00)' OF TYPE 'C' IS NOT A BS2000 PHASE. ERROR CODE '(&01)'  
LMS0180 MEMBER TYPE 'C' ONLY PERMITTED WHEN INPUT AS WELL AS OUTPUT LIBRARY IS A TAPE OR A PLAM LIBRARY  
LMS0181 MEMBER TYPE 'C' ONLY PERMITTED WHEN INPUT LIBRARY IS A TAPE OR PLAM LIBRARY  
LMS0182 MEMBER TYPE 'C' ONLY PERMITTED WHEN OUTPUT LIBRARY IS A TAPE OR PLAM LIBRARY  
LMS0183 MISSING SUBSTATEMENTS WHEN CORRECTING USING UPD STATEMENT  
LMS0184 IDENTIFICATION SPECIFIED IN UPD SUBSTATEMENT DOES NOT EXIST. CHECK ID  
LMS0185 SPECIFIED SEGMENT OF LOAD MODULE TO BE PROCESSED DOES NOT EXIST  
LMS0186 UPDATE TEXT OF MEMBER TO BE CORRECTED NOT CONTAINED WITHIN MEMBER  
LMS0187 CONTROL TEXT OF MEMBER TO BE CORRECTED NOT CONTAINED WITHIN MEMBER  
LMS0188 CROSS CONTROL NUMBER INCORRECT IN UPD SUBSTATEMENT. CHECK CONTROL NUMBER

**Meaning**

E.g. the cross control number >FFFFFFF.

LMS0189 RECORD TYPES NOT EQUAL TO '1' ONLY PERMITTED FOR PLAM LIBRARIES  
LMS0190 NAME DEFINED IN \*NAM SUBSTATEMENT ALREADY EXISTS

**Meaning**

By means of the \*NAM substatement a name was to be defined that already exists in the member or that has already been created using a preceding \*NAM substatement.

LMS0191 REF OPERAND ONLY SUPPORTED FOR PLAM LIBRARIES AND MEMBERS OF TYPE 'R'  
LMS0192 CSECT DEFINED IN A SUBSTATEMENT DOES NOT EXIST IN MODULE. CHECK CSECT NAME  
LMS0193 NAME TO BE RENAMED USING \*NAM SUBSTATEMENT DOES NOT EXIST IN MODULE  
LMS0194 BINARY DEFINITION OF CONTROL TEXT OR UPDATE TEXT NOT PERMITTED IN '\*COR' SUBSTATEMENT OF UPDC STATEMENT  
LMS0195 FILE CANNOT BE ADDED, BECAUSE RECSIZE IS LARGER THAN '(&00)'

LMS0196 MEMBER CANNOT BE EDITED, NUMBERED OR CORRECTED DUE TO '(&00)'

**Meaning**

(&00): RECFORM=F or KEYPOS > 5 or KEYLEN > 16.

LMS0197 ELEMENT CANNOT BE EXTENDED DUE TO '(&00)'

**Meaning**

(&00): RECFORM=F or KEYPOS <> 5 or KEYLEN > 16.

LMS0198 ABSOLUTE CORRECTION ADDRESS TOO HIGH IN UPD SUBSTATEMENT

**Meaning**

In a UPD substatement the sum of base address and correction address > 7FFFFFFF.

LMS0199 WARNING: INVALID RECORD LENGTH

**Meaning**

One or several record(s) of a library member added with fixed record format contain(s) an incorrect record length field not matching the value stored internally. The record with the current record length will be processed.

LMS0200 A PHASE BOUNDED VIA COREIM=NO CANNOT BE UPDATED USING A UPDC STATEMENT

LMS0201 WARNING: ONLY COMPARE-AREA OF RECORDS WILL BE LOGGED

LMS0202 FUNCTION NOT PERMITTED FOR STORED PAM FILE

LMS0203 NO ENTRY EXISTS FOR SPECIFIED REFERENCE CONDITION

**Meaning**

None of the entries in the table of contents matches the specified reference condition (PAR REF=...).

LMS0204 THIS STATEMENT IS ONLY ALLOWED FOR BS2000-PHASES IN PLAM-LIBRARIES

LMS0205 RENAME NOT ALLOWED FOR DELTA MEMBERS

LMS0206 WARNING: PROGRAM ALREADY LOADED

**Meaning**

EDT / EDOR / FMS / PLAM is already loaded.

LMS0207 LIBRARY MEMBERS THAT HAVE FORMAT-B RECORDS CANNOT BE STORED AS DELTA MEMBERS

LMS0208 DELTA MEMBER AND BASE MEMBER MUST NOT BE THE SAME.

**Response**

Define another version for delta element



LMS0209      OVERWRITE=EXTEND NOT ALLOWED WHEN CREATING DELTA MEMBERS  
LMS0210      ERROR WHEN CLOSING LOG MEMBER  
LMS0211      LIBRARY ALREADY EXISTS  
LMS0212      \*END STATEMENT MISSING  
LMS0213      NAME ALREADY EXISTS AS DELTA ELEMENT  
LMS0214      NAME FOR DELTA MEMBER ALREADY EXISTS AS FULL MEMBER  
LMS0215      STATEMENT NOT ALLOWED FOR PRELINKED MODULES WITH COMPLETE 'ESD'

**Meaning**

This statement is not allowed in conjunction with prelinked modules in the new format.

LMS0216      USE \*REP STATEMENT WITH ABSOLUTE ADDRESS FOR PRELINKED MODULES WITH COMPLETE  
              'ESD'

**Meaning**

In the case of prelinked modules in the new format the absolute address within the main module must be used in the \*REP statement.

LMS0217      USE \*COR STATEMENT WITH CSECT NAME AND DISTANCE FOR PRELINKED MODULES WITH  
              COMPLETE 'ESD'

**Meaning**

In the case of prelinked modules in the new format the CSECT name and the distance have to be specified in the \*COR statement, but not the absolute address.

LMS0218      SYNTAX ERROR IN SPECIFIED BASE VERSION  
LMS0219      DELTA MEMBER ONLY ALLOWED FOR PLAM LIBRARIES  
LMS0220      VERSION NOT ALLOWED FOR THIS FUNCTION  
LMS0221      DATE NOT ALLOWED FOR THIS FUNCTION  
LMS0222      NAME LIST NOT ALLOWED FOR THIS FUNCTION  
LMS0223      NO PLAM-LIBRARY  
LMS0224      DELTA AND BASE ELEMENT MUST HAVE SAME TYPE AND NAME  
LMS0225      BUFFER LENGTH < 4  
LMS0226      RECORD LENGTH < 4 OR > 32K

**Meaning**

Record length must be between 4 and 32K

LMS0227 ELEMENT NOT OPENED  
LMS0228 NAME ALREADY EXISTING  
LMS0229 ILLEGAL TID  
LMS0230 ILLEGAL ELEMENT MASK  
LMS0231 NO TOCPRIM/TOCSEC ACTIVE  
LMS0232 UNABLE TO LINK L M S U P  
LMS0233 INVALID CONTROL BLOCK VERSION  
LMS0234 INVALID MAXIMUM LENGTH OF LIBRARY NAME  
LMS0235 MASK ITEM TOO LONG  
LMS0236 INVALID SUBCODE  
LMS0240 WARNING: CLOSING BRACKET WAS ADDED  
LMS0247 SYNTAX ERROR: ILLEGAL MEMBER TYPE DEFINED  
LMS0248 WARNING: OPERAND VALUE OVERWRITTEN BY LAST VALUE SPECIFIED

**Meaning**

In a statement block an operand has been specified more than once. The last value overwrites the former one(s).

LMS0249 UPDATE NOT ALLOWED FOR DELTA-ELEMENTS  
LMS0250 SYNTAX ERROR: ILLEGAL OPERAND VALUE  
LMS0251 PAR PHASE=PK IN NK-WORLD NOT ALLOWED

**Meaning**

Creation of PAMKEY phases in the NON-PAMKEY world is not possible

LMS0252 SYNTAX ERROR: "=" MISSING BEHIND OPERAND  
LMS0253 SYNTAX ERROR: KEYWORD OPERAND MISSING  
LMS0254 SYNTAX ERROR: ILLEGAL KEYWORD SPECIFIED  
LMS0255 SYNTAX ERROR: ABBREVIATION OF OPERAND IS NOT UNIQUE  
LMS0256 INVALID RECORD TYPE

**Meaning**

Record type not 1 - 159 , 163 , 164

LMS0257 THE NAME OF DELTA AND FULL ELEMENTS MUST BE DIFFERENT

LMS0258 INPUT ELEMENT IS EMPTY

LMS0259 PRIMARY AND SECONDARY ELEMENT ARE EMPTY

LMS0260 OUTPUT MEMBER CANNOT BE EXTENDED

LMS0261 MEMBER CANNOT BE EXTENDED DUE TO DIFFERENT '&00)' SPECIFICATIONS

**Meaning**

The member cannot be extended, as the file attributes stored in the member do not match those in the file.

(&00): file attributes (FILETYPE, RECFORM, RECSIZE, KEYPOS, KEYLEN, LOGLEN or VALLEN).

LMS0262 MEMBER CANNOT BE EXTENDED, AS ISAM KEYS ARE STORED IN MEMBER

LMS0263 FILE CANNOT BE ADDED BECAUSE OUTPUT LIBRARY IS NOT A PLAM LIBRARY AND '&00)'

**Meaning**

File cannot be added, as the output library is not a program library and

(&00): PAR KEY=YES or

(&00): the file has the attributes RECFORM=F or KEYPOS>5 or KEYLEN>16

LMS0264 FILE CANNOT BE ADDED BECAUSE '&00)'

**Meaning**

File cannot be added, as the file has either the attributes RECFORM=F, KEYPOS>5 or KEYLEN>16 and PAR KEY=NO.

LMS0265 INTERNAL LMS ERROR WHEN WRITING A MEMBER RECORD

LMS0266 INPUT MEMBER IS NOT A BS2000 PHASE

LMS0302 NO '&00)' (&01' '), INPUT DOES NOT EXIST

**Meaning**

(&00): statement concerned

(&00): member designation.

LMS0303 NO '&00)' (&01' '), NOT IN RANGE OF REFERENCE CONDITION

**Meaning**

(&00): statement concerned

(&01): member designation.

LMS0305 TERMINATION CODE '&00)'. LMS SWITCHES INTO TEST MODE

**Response**

Return to the RUN mode using RST.

LMS0306 SWITCH TO RUN MODE. LIBRARY NO LONGER ASSIGNED

LMS0307 LMS TERMINATION MESSAGE: (&00)

**Meaning**

(&00): value of the job variable(s).

LMS0310 LMS VERSION (&00) LOADED

LMS0311 LMS (&00) (&01' ') ENDED NORMALLY

**Meaning**

(&00): LMS version

(&01): PLAM version (if any).

LMS0312 LMS (&00) (&01' ') ENDED ABNORMALLY

**Meaning**

(&00): LMS version

(&01): PLAM version (if any).

LMS0401 FILE (&00' ') IS LOCKED. ATTEMPT TO BE REPEATED? REPLY (Y=YES; N=NO)

**Meaning**

Y: another attempt will be made

N: no further attempt is made.

LMS0402 MEMBER (&00' ') IN LIBRARY (&01' ') IS LOCKED. ATTEMPT TO BE REPEATED? REPLY (Y=YES; N=NO)

**Meaning**

The member (&00) in the library (&01) is protected against writing or against reading and writing.

**Response**

Y: another attempt will be made

N: no further attempt is made.

LMS0403 TYPE= (&00' ') IN LIBRARY: (&01' ') IS LOCKED. ATTEMPT TO BE REPEATED? REPLY (Y=YES; N=NO)

**Response**

Y: another attempt will be made

N: no further attempt is made.

LMS0411 FILE (&00' ') IS LOCKED. NEXT ATTEMPT STARTED AFTER 6 SECONDS

LMS0412 MEMBER (&00' ') IN LIBRARY (&01' ') IS LOCKED. NEXT ATTEMPT STARTED AFTER 6 SECONDS.

**Meaning**

The member (&00) in the library (&01) is in use, protected against writing or against reading and writing.

LMS0413 TYPE (&00' ') IN LIBRARY (&01' ') IS LOCKED. NEXT ATTEMPT STARTED AFTER 6 SECONDS

LMS0500 INPUT FILE DOES NOT EXIST OR IS NOT ASSIGNED CORRECTLY. AMCB ERROR CODE '(&00)'

LMS0501 WRONG OP-CODE SEQUENCE

LMS0502 INCOMPLETE MODULE (E.G. MISSING END-RECORD)

LMS0503 WRONG RECORD TYPE IN MODULE

LMS0504 OVERWRITE ERROR

LMS0505 THE INPUT FILE IS EMPTY

LMS0506 NO FILE AVAILABLE

LMS0507 BS2000-PHASE IS NOT CORRECT

LMS0508 1ST RECORD IS NO ESD RECORD

LMS0509 PARAMTER OVERWRITE=NO IS SET, BUT A ELEMENT IS EXISTING

LMS0510 ELEMENT OR BASE NOT FOUND

**Meaning**

Input element not found

Output element not found, but OVERWRITE=ONLY set

Base element not found

LMS0511 BASE NOT DELTA-STORED

LMS0512 INPUT AND BASE IN THE SAME CONTAINER

LMS0513 NEITHER LINK NAME NOR FILE NAME EXISTING

LMS0514 TOO MANY LIBRARIES HAVE BEEN OPENED

LMS0515 ILLEGAL ELEMENT-NAME

LMS0516 ERROR ON SEQUENTIAL LIBRARY (AMCB: (&00))

**Meaning**

One of the following errors occurred:

AMCB0040: Wrong BLKSIZE (neither 2048 nor 320 byte)

AMCB0041: Sequence of OUTPUT library sections is wrong

AMCB0042: Rewind beyond actual volume

AMCB0043: Tape restrictions not observed

AMCB0044: Tape and no (LIB)

AMCB0045: File not closed correctly

AMCB0048: Tape mark at reading reverse

AMCB0049: EOV reached during writing

LMS2000 \*\* LLM WARNING \*\* : FUNCTIONALITY IS ASSUMED HOWEVER

LMS2001 INVALID PARAMETERS FOR LLM .

LMS2002 CORRECTION REJECTED BY LLM .

LMS2003 PLAM ERROR DETECTED BY LLM .

LMS2004 LLM INTERNAL ERROR

LMS2005 ILLEGAL LLM MAIN RETURNCODE

LMS2020 ERROR IN LLM OPEN FUNCTION

LMS2021 ERROR IN LLM CLOSE FUNCTION

LMS2022 ERROR IN LLM LIST FUNCTION

LMS2023 ERROR IN LLM UPDATE FUNCTION

LMS2040 OCCURENCE CAN ONLY BE FIRST OR ALL

LMS2050 CHARACTER STRING NOT ALLOWED FOR TYPE L

LMS2100 ILLEGAL LLM SECONDARY RETURNCODE

LMS2101 LLM READING NOT COMPLETE: BUFFER SIZE PHYSICALLY LIMITED.

LMS2102 SPECIFIED PATHNAME NOT FOUND

LMS2103 SPECIFIED SLICE NAME NOT FOUND

LMS2104 SPECIFIED CSECT NAME NOT FOUND

LMS2105 LLM: INVALID IDENTIFICATOR

**Meaning**

Invalid ID passed to LLM

LMS2106 INVALID DISPLACEMENT

LMS2107 LLM: INVALID CONTINUATION

**Meaning**

Invalid continuation flag to LLM

LMS2108 INVALID OLD CONTENT

LMS2109 INVALID LENGTH

LMS2110 LLM: INVALID MODE

**Meaning**

INVALID OPEN MODE PASSED TO LLM

LMS2111 LLM PARAMETERS HAVE INVALID VALUES

LMS2112 INVALID PATHNAME SPECIFIED

**Supplementary information**

The following supplementary information appears in various combinations in addition to the actual text of the LMS messages:

supplinfo	Meaning
(STATEMENT MEMBER INPUT)	Statement input from member
(COP OR DUP)	COP or DUP function
(DATA CARD INPUT)	Error in CAT function
(DATA CARD OUTPUT)	Error in PCH, LAP function
(FIRST STATEMENT IS // ...) // ...)	The first statement is // ...
FUNCTION TERMINATED	Function is aborted
(LIBRARY INPUT)	Library input
(LIBRARY OUTPUT)	Library output
(LISTING-MEMBER-OUTPUT)	Log output to member
(LAST STATEMENT IS NOT // MENM OR MEND	The last statement is not // MENM or // MEND
(SECOND STATEMENT IS NOT // JOM OR // JOB)	The second statement is not // JOM or // JOB
STATEMENT(S) IS (ARE) SKIPPED	The entire statement is ignored
SYSIPT (CATALOG- OR CORRECT-FUNCTION)	SYSIPT for reading data and correction cards
SYSLST (LISTING)	Log output to SYSLST
SYSOPT (PUNCH-FUNCTION)	SYSOPT for punch function
ILLEGAL COMMAND	Impermissible command
ILLEGAL OPERAND	Impermissible operand
ERROR FROM COMMAND	Error attributable to illegal command
WHOLE ITEM IS SKIPPED	Operands are skipped up to the next comma



supplinfo	Meaning
OUTPUT-LIBRARY LOCKED	The output library is locked
OPENERROR ON LIBRARY/ELEMENT	Error occurred when library/member was opened
OUTPUT-LIBRARY MISSING	The output library does not exist
OUTPUT-ELEMENT CHANGED	The output member to be created has a different version number
PAR KEY=NO AND RECFORM=F	Processing operand KEY=NO is set and the file has a fixed record format
PAR KEY=NO AND KEYPOS>5	Processing operand KEY=NO is set and the KEYPOS value of the file to be added is greater than 5
PAR KEY=NO AND KEYLEN>16	Processing operand KEY=NO is set and the KEYLEN value of the file to be added is greater than 16
PAR KEY=YES AND OSM-LIBRARY	Processing operand KEY=YES is set and the output library is an old source or macro library
RECFORM=F AND OSM-LIBRARY	The file to be added has records of fixed length and the output library is an old source or macro library
KEYPOS>5 AND OSM-LIBRARY	The file to be added has a KEYPOS greater than 5 and the output library is an old source or macro library
KEYLEN>16 AND OSM-LIBRARY	The file to be added has a KEYLEN greater than 16 and the output library is an old source or macro library
RECFORM=F	The library member has records of fixed length
KEYPOS>5	The library member has been added using PAR KEY=YES and KEYPOS is greater than 5
KEYLEN>8	The library member has been added using PAR KEY=YES and KEYLEN is greater than 8 (applies only when calling EDT)
KEYLEN>16	The library member has been added using PAR KEY=YES and KEYLEN is greater than 16 (applies only when calling EDOR)
PAR KEY=YES	Processing operand KEY is set to YES
KEYS DO EXIST IN ELEMENT	The library member to be extended has been added using PAR KEY=YES and contains the ISAM keys
DIFFERENT FILETYPE/VALUE PROPAGATION (MIN/MAX)	The file type or the "VALUE PROPAGATION" of the file to be added does not match that of the library member to be extended

## Messages

---

supplinfo	Meaning
DIFFERENT RECORD FORMAT	The record format of the file to be added does not match the record format of the library member to be extended
DIFFERENT RECORD SIZE	The record length of the file to be added does not match the record length of the library member to be extended
DIFFERENT KEYPOSITION	The position of the keys of the file to be added does not match the position of the keys of the library member to be extended
DIFFERENT KEYLENGTH	The length of the keys of the file to be added does not match the length of the keys of the library member to be extended
DIFFERENT LOGLENGTH	The LOGLENGTH of the file to be added does not match the LOGLENGTH of the library member to be extended
DIFFERENT VALUE LENGTH	The VALLEN value of the file to be added does not match the VALLEN of the library member to be extended
OUTPUT-LIBRARY IS NOT A PLAM-LIBRARY	The output library is not a program library
FIXED RECORD FORMAT- ON INPUT-FILE	The library member to be extended contains no file attributes and the input file has fixed-length records
KEYPOSITION 5 ON INPUT-FILE	The library member to be extended contains no file attributes and the input file has a key position which is not equal to 5
KEYLENGTH>16 ON INPUT-FILE	The library member to be extended contains no file attributes and the input file has a key length greater than 16
RECORD SIZE>2032 ON INPUT-FILE	The file to be added has a record length of more than 2032 bytes
(COMMAND INPUT/ TYP J)	Statement input from a J-type member

## System queries

The following queries may be output by LMS. The user must answer them with Y or N.

```
DO YOU WISH A BKPT (Y/N) ?
```

### Meaning

When job switch 31 is set and a program error occurs in interactive mode, LMS asks whether a BKPT macro should be issued.

```
LMS0403 TYPE ... IN LIBRARY ... IS LOCKED. ATTEMPT TO BE REPEATED?
REPLY (Y=YES; N=NO)
```

### Meaning

The member type is locked against writing.

```
LMS0402 MEMBER ... IN LIBRARY ... IS LOCKED. ATTEMPT TO BE REPEATED?
REPLY (Y=YES; N=NO)
```

### Meaning

The member is locked against reading and writing.

## Access method messages

The messages issued by the internal LMS access methods have the following format:

$\left. \begin{array}{l} \text{AMCB} \\ \text{PLAM} \end{array} \right\} : \text{xxxx} ** \text{DMS: } \text{yyyy}$
---

where

xxxx is the AMCB/PLAM error key.

yyyy is the DMS error code (see system messages).

The error codes from 200 onwards are PLAM error codes. These PLAM error codes can also be queried using command

```
/HELP PLAxxxx
```

## Messages

---

xxxx	Meaning of AMCB error codes
0000	No error
0001	DMS error (for error code see RET3)
0002	Illegal op code
0003	File name missing in control block
0004	No / modified FCB address, or FCB address for FOP with DMS-OPEN points to active FCB
0005	Incorrect op code sequence
0006	Library type invalid
0007	Contradictory LIB types in control block
0008	Library has been repaired
0009	Library must be repaired
0010	Address not within member limits
0011	On writing: library limit reached
0012	Contradictory information in control block and FCB
0013	Supplementary information missing
0014	Invalid supplementary information
0015	Record too long
0016	Illegal compression flag in directory
0017	Last member in library has been deleted
0019	This is not an LMS library.
0020	TOC area overflow of R-library DIR1 > 30 PAM pages
0021	TOC area overflow of R-library DIR2 for this module > 4 PAM pages (too many CSECTS/COMMONS/ENTRYs)
0022	Module not complete (e.g. no END record)
0023	Incorrect record type in module
0024	Warning: library overflow imminent
0025	No DIR2 entry provided
0026	OSM library must be reorganized by means of the /VERIFY command
0040	Incorrect block size on the input library tape, neither 2048 nor 320
0041	Sequence of library sections incorrect for output tape library
0042	With continuation tape processing for input tape libraries, positioning to a previous reel is required but not permitted
0043	On the input tape library there is a tape mark in conjunction with *Start block
0044	Tape library and no short designation in the LIB statement
0045	The tape input library was not properly closed
0046	No standard labels for tape input libraries
0047	Reserved
0048	The BOT marker was sensed during positioning on the input tape library
0049	End of tape was reached when writing the output library tape
0050	Overwrite error
0051	Insufficient memory
0052	Member has been overwritten
0053	Input file is empty
0054	Empty file is replaced
0062	Function not implemented
0063	No files present
0064	FMS cannot be loaded
0065	XFR/PHASE record not present
0066	First record not an ESD record

xxxx	Meaning of AMCB error codes
0100	Illegal program file
0101	DMS error
0102	Unknown file type
0107	Neither file name nor link name entered
0108	User error
0109	Open error
0111	No free space in file table
0112	FSTAT error
0118	No empty file for CREATE
0119	Open for empty file
0120	File name invalid
0121	File ID has no entry in file table
0122	Requested open status different from actual status
0123	No further space for FCB
0124	No further space for access indicator in file table
0125	Second access to output library
0126	No further space for link table entry
0127	Link table entry missing
0129	VSN check reveals error
0131	Library still open for CTL or PRT
0134	CTL member still open
0135	Link name LIBxxx - xxx is not numeric
0136	Access error, e.g. file locked
0137	Second access to a sequential library
0150	Access method not known

## Messages

---

xxxx	Meaning of PLAM error codes
	** 200 - 299 Warnings **
0201	Library does not exist
0202	No library name in the PLCB and no /FILE command
0203	No PLAM library
0204	Library not new
0205	Member does not exist
0206	Member already exists
0207	Variant does not exist
0208	EOF
0209	Record not found
0210	No record transferred
0211	End of record type
0212	No further entry in directory
0214	Format B record truncated
0216	The record that has been read is shortened
0217	Secondary allocation of the library is set to 0; the library cannot be extended automatically
0219	Access to the library via RFA with SHAREUPD=YES not allowed
0221	Format B record too short
0222	Format B record too long
0255	SVC PLAM not available
0290	Computer center exit routine request rejected
	** Multiple access restrictions **
0301	Member opened for different PLCB
0302	Write access to member type
0303	Member type locked
	** 401 - 599 User program errors **
0401	Illegal PLCB address
0402	Illegal operands / PLCB
0403	Illegal function
0404	ATTACH not possible for a PLCB
0405	Library not assigned for PLCB
0406	Member opened for PLCB
0407	Neither link name nor library name in PLCB
0408	Incorrect P2-PLCB address
0409	Illegal P2-PLCB
0410	Illegal open mode
0411	Address of data area missing
0412	Member for PLCB not opened
0413	Invalid variant number
0414	Variant number overflow
0415	Member not opened for WRITE/UPDATE/EXTEND
0416	Member not opened for INPUT/UPDATE/EXTEND
0417	Inconsistency in data area
0418	Member designation 1 equal to member designation 2
0419	Illegal data area
0420	Illegal length of data area
0421	Illegal block size
0422	Illegal record type end condition
0423	Illegal ATTACH mode
0424	PLCB assigned for INPUT

xxxx	Meaning of PLAM error codes
0425	INPUT assignment for destroyed library
0426	Illegal ATTACH condition
0427	Illegal OPEN condition
0428	Invalid entry for KEEP
0429	Illegal RENAME condition
0430	Illegal member type
0431	Illegal member name
0432	Invalid version entry
0433	Invalid entry for INQUIRY
0434	Invalid entry for SECURITY-ERASE
0435	Old member type not equal to new member type
0436	RENAME error
0437	Illegal record length
0438	Invalid record number
0439	Illegal FCB type in the /FILE command
0440	SHARUPD=NO not permitted
0441	Invalid entry for OPEN in the /FILE command
0442	Illegal record type
0443	No member type locked for PLCB
0444	Member type locked for PLCB
0445	Invalid entry for variant selection
0446	Invalid entry for version selection
0447	Invalid member access ID
0448	Invalid user date identifier
0449	Illegal operand for GWRK
0450	Illegal operand for RWRK
0451	Invalid work area address
0452	Invalid work area length
0453	Illegal secondary name
0454	Invalid entry for secondary name selection
0455	Invalid format B buffer address
0456	Invalid format B buffer length
0457	Illegal format B buffer
0458	Invalid entry for format B positioning
0459	Invalid attribute name
0460	ATTACH mode EXCE not for P1
0461	Invalid option for further search under \$TSOS
0462	Invalid option for system library
0463	Invalid catalog designation
0464	Invalid option for format B record transfer
0465	Invalid option for format B record masking
0466	Invalid format B record mask
0467	Invalid search default
	** 601 - 699 Library data invalid/destroyed
0601	Inconsistency in the member description
0602	Illegal format A control area
0603	Inconsistency in the format A control area
0604	Inconsistency in the format A control area entry
0605	Inconsistency in the format A data area
0606	Inconsistency in the directory
0607	Illegal operation code for restart
0608	Illegal restart block

## Messages

---

xxxx	Meaning of the PLAM error codes
0609	Inconsistent disk storage management
0610	Inconsistent member name
0611	Inconsistency in the variant catalog
0612	Inconsistent format B control area ** 701 - 799 System service
0701	ENASI macro error
0702	DISSI macro error
0703	ENQAR macro error
0704	DEQAR macro error
0705	ENAMP macro error
0706	REQMP macro error
0707	RELMP macro error
0708	OPEN macro error
0709	PAM macro error
0710	CLOSE macro error
0711	FILE macro error
0712	EXRTN macro error
0713	FSTAT macro error
0714	REQM macro error
0715	RELM macro error
0716	\$REQM macro error
0717	\$RELM macro error
0718	\$NCREBO macro error
0719	\$NDESBO macro error
0720	\$NENQBO macro error
0721	\$NDEQBO macro error
0722	Error on STAM macro
0723	Error on RDTFT macro
	** Internal PLAM errors **
0801	Illegal LINK P2-PLCB → P1-PLCB
0802	Lock set for end of function
0803	Control area opened for access
0804	Control area not open for access
0805	Search error in control area
0806	Illegal compression mode
0807	Library table not locked
0808	Library not locked
0809	Member not locked
0810	A library for PLCB has already been opened
0811	No library opened for PLCB
0812	Close library but member is open
0813	Member for PLCB already opened
0814	No member opened for PLCB
0815	Illegal open mode for member
0816	Invalid index flag in TOC area
0817	Disk storage management is not locked
0818	Restart memory not locked
0819	Library table header not assigned
0820	Library table header assigned
0821	Error in lock sequence
0822	Invalid name identifier
0823	Key already exists



xxxx	Meaning of the PLAM error codes
0824	Error during allocation of data area
0825	Member type too long
0826	Member name too long
0827	Member version too long
0828	Key in directory not available
0829	No buffer area available for directory
0830	Directory overflow
0831	Inconsistencies in I/O management
0832	No buffer area available
0833	Buffer address not found
0834	Buffer area not reserved
0835	Buffer management not initialized
0836	Illegal PRIV-FLAG for \$REQM
0837	Illegal ACCESS-FLAG for \$VALD
0838	Invalid number of bytes
0839	Illegal memory class
0840	Storage not available
0841	Storage area not released
0842	Invalid storage address
0843	Part of page not allocated
0844	Attempt to open an open file
0845	Attempt to close a closed file
0846	Illegal open mode
0847	Library file not open
0848	Invalid number for the I/O request
0849	Illegal LOCK/WAIT identifier
0850	Bourse already generated
0851	Bourse not generated
0852	Lock status error of library table
0853	Lock status error of library
0854	Lock status error of disk storage
0855	Lock status error of member
0856	Library does not exist
0857	Library is empty
0858	Input/output not terminated
0859	Library access ID changed
0860	Member not found
0861	No valid storage area
0862	Invalid directory number
0863	Secondary name too long
0864	Variant number too long
0865	Search error in format B control area
0866	Format B record access not possible
0867	Format B record access possible
0868	Attribute name too long
0869	Invalid option for parallel update
0870	Invalid FCB address

## Messages

---

xxxx	Meaning of the PLAM error codes
1001	Address of system version entry invalid
1002	TASK WORK AREA could not be created
1003	TASK WORK AREA destroyed
1004	Register 1 not equal to 0
1005	Incompatible PLAM version
1006	Member was opened for WRITE, UPDATE or EXTEND
1007	Member no longer exists
1008	Variant was changed
1009	Type lock was set
1010	Invalid link between P1-PLCB and PLCB
1011	Too many calls from RESTART
9999	Unknown error

## Appendix

### Conversion of MLU, LMR, COBLUR to LMS

LMS incorporates the functions of MLU, LMR and COBLUR. It can also access libraries created by these programs. The following points should be noted:

#### **MLU format**

Libraries created by MLU are processed by LMS. Since members processed by MLU do not contain a version number, LMS assumes the version number 0 (zero) for such members. Macro and source libraries created by LMS have the MLU format and can be further processed by MLU. (Program libraries cannot be processed by MLU.)

#### **LMR format**

Libraries created by LMR can be processed by LMS. Since members processed by LMR do not contain a version number, LMS assumes the version number 0 (zero) for such members. Object module libraries created and processed by LMS (and which are not empty) can be further processed by LMR. (Program libraries cannot be processed by LMR.)

#### **COBLUR format**

Libraries created by COBLUR can only be read by LMS. Changes cannot be made to such libraries. Since the members of a COBLUR library contain neither version number nor date, zero is assumed for both. LMS ignores the organization into library divisions. It is therefore not possible to determine in which divisions the members are stored by reference to the log of a directory. If identically named members are stored in a number of divisions, these names appear a corresponding number of times in the directory log. If a name that exists in more than one division is specified using LST, the first member found will appear in the log.

Similarly when DUP is used, only the first member found will be duplicated. First, identically named members must be renamed with COBLUR. The entire library can then be duplicated.

## BS1000-BS2000 compatibility

Using LMS, libraries on tapes can be transported from BS1000 to BS2000 and vice versa.

LMS can process BS1000 tape libraries having the old (320 bytes) or the new (2 KB) block size. The sequential libraries created by LMS, however, always have the new block size (2-KB blocks).

If a tape library created in BS2000 is subsequently to be processed in BS1000, ACCESS-METHOD=BTAM must be specified in the /SET-FILE-LINK command.

A BS2000 program file that is to be transferred to a BS1000 library tape (ADDC), must be linked with COREIM=N in the PROGRAM statement of TSOSLINK.

Standard practice in BS1000 is for tape files to have the file name DOS-LIB unless some other name is specified by way of VOL and TPLAB cards when the tape is created.

When assigning member names, it should be noted that names of BS1000 job macros and modification records must not exceed 7 characters in length. Job macro names must be prefixed by the character "." (period) and modification records for job macros by the character "/" (slash).

### Processing of continuation tapes

Continuation tapes created under BS1000, can also be processed under BS2000. However, they must be assigned individually with different short designations in the /SET-FILE-LINK command, e.g. tape 1 with LIB001, tape 2 with LIB002, etc. Since they have the same file name, the first input library must be closed and the file name erased from the catalog before the 2nd tape is assigned.

### Special considerations for the storage of BS1000 job macros

Job macros can be entered in source program libraries by means of the WRT function. The input can only be made from the display terminal; inputs via SYSDTA/SYSIPT are not possible under BS2000. Job macros are identified by virtue of the fact that the first character of the name is a period.

LMS checks that job macros observe the following conditions:

1. The first statement must not begin with //. It must be a load operand statement (see Monitor description in the "JMS1" manual [11]) or be blank.
2. The second statement must begin with // JOM or // JOB.
3. The last statement must begin with // MENM or // MEND.

If these rules are not observed, a job macro will not be cataloged by LMS.

In BS1000 job macros, NOP within the LMS statements reserves space for modifications.



## Statements and processing operands

The following statements are supported for reasons of compatibility only.

- CAT      Add members via SYSIPT
- COP      Copy members
- LAP      List and output to SYSOPT
- LIBIN    Assign input library
- LIBOUT   Assign output library
- PCH      Output members to SYSOPT
- WRT      Add members to libraries via SYSOPT

**LIBIN Assign input library**

LIBIN is supported for reasons of compatibility only, since its functions are now covered by LIB.

This statement defines the implicit input library and closes the previous input library.

Operation	Operands
LIBIN	$\left\{ \begin{array}{l} (\text{lib}) [ , (\text{vsn}) ] \\ \text{libname} \\ \text{LINK=linkname} \end{array} \right\}$

LIBIN	Statement name.
lib	Short designation of the input library.
vsn	Volume serial number (6 characters).
libname	Fully qualified file name of the input library.
linkname	Link name referring to the input library.

**LIBIN**

- closes the previous input library
- cancels the definition of the implicit input library
- defines as the input library that library specified by "libname", "lib" or "linkname".

The implicit input library thus defined remains valid until the next LIBIN. At the start of the LMS run and following a LIBIN with no operands, the implicit input library is undefined.

Following errors, the implicit input library is undefined and must be assigned anew.

When a short library designation (lib) or a link name (LINK=...) is used in LIBIN, a /SET-FILE-LINK command must be given (see page 38).

A volume serial number may be specified by means of the "vsn" operand. This must match the VSN of the assigned volume.

A sequential library cannot simultaneously serve as input and output library.

## LIBOUT Assign output library

LIBOUT is supported for reasons of compatibility only, since its functions are now covered by LIB. However, LIBOUT must still be used if a sequential library is to be defined as the output library.

This statement closes the previous output library and defines the new output library.

Operation	Operands
LIBOUT	$\left[ \left\{ \begin{array}{l} (\text{lib}) [ , \left\{ \begin{array}{l} (\text{vsn}) \\ \text{NEWLIB} \\ \text{NEWLIB} (\text{vsn}) \end{array} \right\} ] \right\} \right. \\ \left. \begin{array}{l} \text{libname} \\ \text{LINK=linkname} \end{array} \right\} \right]$

LIBOUT	Statement name.
lib	Short designation of the output library.
vsn	Volume serial number (6 characters).
NEWLIB	Creation of new library.
libname	Fully qualified file name of the output library.
linkname	Link name referring to the output library.

### LIBOUT

- closes the previous output library
- cancels the definition of the implicit output library
- defines as the output library that library specified by "libname", "lib" or "linkname".

The implicit output library thus defined remains valid until the next LIBOUT. At the start of the LMS run and following a LIBOUT with no operands, the implicit output library is undefined.

Following errors, the implicit output library is undefined and must be assigned anew.

A volume serial number may be specified by means of the "(vsn)" operand. This must match the VSN of the assigned volume.

When a short library designation (lib) or a link name (LINK=...) is used in LIBOUT, a /SET-FILE-LINK command must be given (see page 38).



A sequential library cannot simultaneously serve as input library and output library. LIBOUT causes the current output library to be closed. This same library can then be assigned as the input library.

If a tape is assigned for "lib", a new tape library will always be opened, i.e. new header labels are written for the library.

For sequential libraries, NEWLIB must always be specified in LIBOUT.

The specification NEWLIB is permitted only for a file that is to be newly created; it causes the output library to be set up as an empty library. The following attributes must be given for the file in the FILE command:

Library type	FCBTYPE	KEYPOS	KEYLEN
Source library	ISAM	5	8
Macro library	ISAM	5	8
Module library	PAM		
Sequential library	BTAM		

In the case of an ISAM file, its type is defined as being a macro library or a source library the first time it is used as an output library.



### Processing operands

The following processing operands are supported for reasons of compatibility only.

- PAR COLLECT Collect statements
- PAR SAVE Save members during correction

### PAR DECOMPRESSED

Control compression for macros and source programs

The processing operand DECOMPRESSED is supported for reasons of compatibility only. For program libraries, it has the same effect as NOP. For other libraries, it controls compression for the output of macros and source programs to libraries.

Operation	Processing operand
PAR	$DEC[OMPRESSED] = \left[ \begin{array}{c} YES \\ NO \\ ? \end{array} \right]$


- YES All members are output in decompressed form.
- NO All members are output in compressed form.
- ? The current value is logged.

---

## References

- [1] **BS2000 System User**  
Ready Reference
- Target group*  
Experienced BS2000 users.
- Contents*
- An overview of
- commands and macros in BS2000
  - instructions and Assembler statements
  - statements for the software products and utility routines
    - EDT, EDOR, SORT, LMS, ARCHIVE, PERCON, LEASY
    - TSOSLNK, DCAT, PASSWORD, FDEXIM, FDRIVE, DPAGE, SODUMP, TCOMP2, PRSERVE
  - principal tables and registers of BS2000
  - code tables
  - system conventions
- Application*
- BS2000 interactive and batch modes.
- [ 2] **BS2000 Binder-Loader-Starter (BLS)**  
User Guide
- Target group*  
Software developers
- Contents*
- The binder-loader-starter (BLS) system consists of the following functional units:
- Linkage editor BINDER
  - Dynamic binder loader DBL
  - Static loader ELDE
- The various sections contain functional descriptions and examples, plus a reference section with statements, commands and, where applicable, macros.

- [ 3] **BS2000  
Utility Routines**  
User Guide
- Target group*  
BS2000 users (non-privileged)
- Contents*  
Utility routines for non-privileged BS2000 users
- Applications*  
BS2000 timesharing mode
- [ 4] **EDOR (BS2000)**  
Reference Manual
- Target group*  
– Data entry operators  
– Programmers
- Contents*  
Description of the statements to the EDOR File Editing System
- Applications*  
BS2000 interactive mode
- [ 5] **EDT (BS2000)**  
**Statements**  
User Guide
- Target group*  
– EDT newcomers  
– End users
- Contents*  
– Processing of SAM and ISAM files and elements from program libraries  
– Introduction to the basic principles of EDT and description of the operating modes  
– Creation of EDT procedures  
– Descriptions of all the EDT statements. Frequent applications are illustrated with the aid of numerous examples.
- Applications*  
File editing

- 
- [ 6] BS2000  
**Introductory Guide for System Users**  
User's Guide
- Target group*  
BS2000 users
- Contents*
- Introduction to BS2000
  - Description of the most frequent user commands
  - Introduction to using the utility routines and software products EDT, SORT, ARCHIVE, TSOSLNK, LMS and PERCON
  - Notes for the programmer
- Applications*  
BS2000 interactive mode and batch mode
- [ 7] BS2000  
**User Commands (SDF Format)**  
User Guide
- Target group*  
BS2000 users
- Contents*  
BS2000 user commands in the syntax of the dialog interface SDF (System Dialog Facility)
- Applications*  
BS2000 interactive/batch mode with SDF
- [ 8] BS2000  
**Executive Macros**  
User Guide
- Target group*
- BS2000 assembly language programmers (non-privileged)
  - System administrators
- Contents*
- All Executive macros in alphabetical order with detailed explanations and examples; selected macros for DMS and TIAM
  - Macro overview according to application areas
  - Comprehensive training section dealing with eventing, serialization, inter-task communication, contingencies
- Applications*  
BS2000 application programs
- 

## References

---

- [ 9] BS2000  
**Laser Printer**  
Reference Manual  
BS2000 application programs
- [10] BS2000  
**Software Produce FMS**  
User's Guide
- [11] BS1000  
**Systems for Automatic Job Management (JMS)**  
Reference Manual Part 1
- [12] **Systems Standards** (BS1000, BS2000, TRANSDATA, PDN)  
Reference Manual  
*Target group*  
Users of Siemens mainframes  
*Contents*
  - Operating system standards for BS1000, BS2000 and TRANSDATA PDN
  - Standards for data volumes
  - Codes for character representation
- [13] **LMS** (BS2000)  
Pocket Reference Guide  
*Target group*  
Users familiar with LMS  
*Contents*  
Overview of the statement formats and processing operands, accompanied by brief functional descriptions
- [14] BS2000  
**DMS Introductory Guide and Command Interface**  
User Guide  
*Target group*  
Non-privileged BS2000 users  
*Contents*
  - Functions of DMS in BS2000
  - Processing of disk and tape file
  - Access methods UPAM, SAM, BTAM, EAM, ISAM
  - DMS commands

---

[15] **LMS (BS2000)**  
**Subroutine Interface**  
User Guide

*Target group*

- LMS users
- Programmers

*Contents*

Overview of the possible applications, call preparations and a description of the subroutine functions. The subroutine interface is offered for COBOL, C and Assembler. For each of these programming languages, the parameter structure is described and an example provided.

**Ordering manuals**

The manuals listed above and the corresponding order numbers are to be found in the *List of Publications* issued by Siemens Nixdorf Informationssysteme AG, which also tells you how to order manuals. New publications are listed in the *Druckschriften-Neuerscheinungen (New Publications)*.

You can arrange to have both of these sent to you regularly by having your name placed on the appropriate mailing list. Your local office will help you.





## Index

\$ statement 201  
\*BAS correction statement for load modules 185  
\*BAS correction statement for object modules 168  
\*CHANGE correction statement for text members 114  
\*CON correction statement for load modules 185  
\*CON correction statement for object modules 168  
\*COR correction statement for load modules 186  
\*COR correction statement for object modules 169  
\*DEL correction statement for load modules 188  
\*DEL correction statement for object modules 171  
\*DEL correction statement for text members 113  
\*END correction record for object modules 172  
\*END correction statement for load modules 188  
\*END correction statement for text members 116  
\*ID correction statement for load modules 188, 195  
\*ID correction statement for object modules 172  
\*INS correction statement for object modules 173  
\*INS correction statement for text members 112  
\*INV correction statement for object modules 174  
\*NAM correction statement for object modules 175  
\*REM correction statement for load modules 189, 195  
\*REM correction statement for object modules 175  
\*REP correction statement for object modules 176  
\*REP correction statement for text members 113  
\*SEG correction statement for load modules 189  
\*SET correction statement for object modules 178

### A

abnormal termination 246  
activating run mode 248  
activating test mode 248  
ADD statement 91  
    format 1 93  
    format 2 96  
    format 3 97

- format 4 100
- format 5 103
- adding comparison statistics 159
- adding data to a library 91
- adding delta members 60
- adding files 93
- adding load modules 103
- adding member records from the LMS statement stream 100
- adding members 43
- adding members from an FMS library 97
- adding object modules 96
- adding source programs 253
- AMCB error codes 318
- archive library 20
- assembling source programs 253
- assign library temporarily 39
- assigning libraries 38, 135
- assigning sequential libraries 41, 137
- attributes of a library 11

### **B**

- base address, definition 168, 185, 209
- BASE processing operand 209
- BLKCTRL 75
- blocked format 84
- branching to a user program 285
- branching to user programs 196
- BS1000 job macros 326
- BS1000 phases 20, 103
- BS1000 tape libraries 326
- BS2000 phases, correction 183

### **C**

- calling procedures 56
- cancelling corrections 189, 195
- changing records 114
- check field 48
  - definition 210, 238
  - numbering 252
  - string definition 243
- CHECK processing operand 210
- closing libraries 138
- COBLUR 325
- columns, defining number of 227
- COM statement 105

- comments 83
- compare function, control 212
- COMPARE processing operand 212
- comparing members 50, 105, 262, 274
- comparison base 59
- comparison field 212
- comparison log 50, 105, 213
- comparison result 51
- comparison statistics 51, 105, 213
  - addition 159
  - creation 245
  - deletion 160
  - output 159, 282
  - storage 158
- compatibility, BS1000-BS2000 326
- compiler result information 24
- construction specification 34
  - examples 34
  - symbols 34
- container 16, 58
- contents of a member 22
- continuation characters 84
- continuation lines 83
- continuation tapes 20, 326
- control number 70
- control of screen overflow 68
- control section attributes, modification 178
- control statement input 117
- controlling form feed 234
- controlling log output 66
- controlling physical deletion 215
- controlling statement input 68
- controlling the compare function 212
- controlling the LMS run 63
- controlling the log output 148
- controlling the output format for directories 249
- converting corrections 174
- converting lowercase to uppercase type 225
- converting REP records 174
- converting text corrections 174
- COR statement 53, 108
- correcting a source program using COR 269
- correcting an object module using UPD 272
- correcting delta members 62

- correcting files 132
- correcting load modules 54, 183
- correcting members 53
- correcting object modules 54, 165
- correcting source programs 253
- correcting text members 53, 108, 127
- correcting text records 169, 186
- correction, cancellation 195
- correction input, termination 116, 188
- correction journal record 174
  - deletion 188
- correction statements, termination 172
- correction statements for COR 109
- correction statements for UPDC 183
- correction statements for UPDR 165
- correction statements from comparison log 53, 213, 274
- corrections
  - cancellation 189
  - conversion 174
  - remove 175
- creating comparison statistics 245
- creating files 132
- creating text members 127
- cross control number, definition 168, 185
- CTL statement 117

## D

- data, adding to a library 91
- deactivating test mode 248
- deactivating the user exit 197
- DECOMPRESSED processing operand 332
- define identification 172
- defining a base address 168, 185, 209
- defining a segment 189
- defining a string in the check field 243
- defining an identification 188, 195
- defining reference conditions 239
- defining segments 240
- defining the check field 210, 238
- defining the cross control number 168, 185
- defining the FCB type 217
- defining the number of columns 227
- defining the number of lines 227
- defining the record format 219

- defining the scope of output 221
- DEL statement 118
- deleting comparison statistics 160
- deleting correction journal records 188
- deleting delta members 61
- deleting members 47, 118
- deleting object module parts 171
- deleting records 113
- delta as organizational aid 59
- delta as storage form 59
- delta member 58
  - adding 60
  - locking 62
  - organization 59
  - storage 59
- delta members
  - correcting 62
  - deletion 61
  - overwriting 62
  - processing 266
  - renaming 62
- delta method 5, 58
- delta quantity 60
- delta sequence 59
- delta storage method 58
- delta structure 59
- delta tree 59
- delta trees, duplicating 124
- DESTROY processing operand 215
- diagnostic aids 73
- directory 11
  - output 55, 162
  - sort 242
- directory of a library 11
- displaying assigned libraries 139
- displaying members 132
- DSDD record 171
- dummy function 144
- DUP statement 120
  - format 1 121
  - format 2 124
- duplicate with structure 124
- duplicating delta trees 124
- duplicating members 46, 121, 258

dynamically loading the user program 197

**E**

edited data 23  
editor run 129  
EDR statement 126  
    format 1 127  
    format 2 132  
    format 3 132  
EDT statement 126  
    format 1 127  
    format 2 132  
    format 3 132  
effect of processing operands 63  
element 5, 11  
END statement 133  
entry address LMSUP 289  
ERRCONS processing operand 216  
error handling 246  
error messages 293

**F**

FCB type, definition 217  
FCBTYPE processing operand 217  
file  
    correction 132  
    creation 132  
file attribute BLKCTRL 75  
file attributes, transfer 224  
files, adding 93  
filing members using the delta method 58  
FMS library 97, 156  
form feed, control 234  
formal comparison 50, 105, 212  
format of statements 82  
FORMAT processing operand 219  
functions of LMS 5

**G**

generating ISAM files 154

---

**I**  
identification, definition 172, 188, 195  
IFG format masks 24  
IFG user profile 24  
INCLUDE record 171  
    insertion 173  
increment of numbering 252  
INFO processing operand 221  
initial value of numbering 252  
input library 12  
inserting a REP record 176  
inserting an INCLUDE record 173  
inserting records 112  
interface, user exit 198  
interrupting the LMS run 71  
INTR command 71  
INV correction statement  
    format 1 174  
    format 2 174  
ISAM, SAM and PAM files 24  
ISAM file 79  
    generation 154  
ISAM key, transfer 224  
ISD record 171  
issuing system commands 160

**J**  
job switch 66  
job switches, using 74

**K**  
KEY processing operand 224

**L**  
LCASE processing operand 225  
leave test mode 150  
LIB statement 134  
    format 1 135  
    format 2 138  
    format 3 139  
LIBIN statement 329  
LIBOUT statement 330  
libraries, type-related 18  
libraries required 90  
library 5, 11

- closing 138
- display 139
- short designation 39
- library assignment 38, 135
- library formats 6, 13
- LINE processing operand 227
- lines, defining number of 227
- link name 38, 39
- list members 23
  - printing 140
- listing members 46, 140
- LLMs 24
- LMR 325
- LMS functions 5, 37
- LMS in batch mode 9
- LMS in interactive mode 9
- LMS log 66, 148
- LMS queries 317
- LMS run
  - controlling 63
  - interrupting 71
- LMS termination 133
- LMSUP 289
- load module 24
- load modules
  - addition 103
  - correction 54, 183
- locking delta members 62
- log output
  - control 148
  - controlling 66
- LOG processing operand 228
- log statements 228
- logical comparison 50, 105, 212
- logical deletion 47, 61
- lowercase/uppercase conversion 225
- LSD record 171
- LST processing operand 229
- LST statement 140



**M**

- macro library 18, 28
- macros 22
- member 5, 11
  - assign as system input file 56
  - comparison 50, 262, 274
  - duplication 258
  - listing 46
  - output 277
- member contents 22, 28
- member correction 53
- member designation in program libraries 25
- member designation in sequential libraries 21
- member designation in type-related libraries 29
- member processing 42
- member record, numbering 47
- member records, numbering 145, 252
- member records from the LMS statement stream, addition 100
- member type 14, 22, 28
  - predefinition 251
- member type C 24
- member type D 24
- member type F 24
- member type H 24
- member type J 23
- member type L 24
- member type M 22, 28
- member type P 23
- member type R 22, 28
- member type S 22, 28
- member type U 24
- member type X 24
- member types per statement 88
- member version 15
- members
  - adding 43
  - adding from an FMS library 97
  - comparison 105
  - deleting 47, 118
  - display 132
  - duplicating 46
  - duplication 121
  - listing 140
  - output to files 152

- output to FMS library 156
- outputting 46
- overwriting 235
- renaming 55, 142
- messages 293
  - access methods 317
  - output 216
  - supplementary information 313
- MLU 325
- modifying control section attributes 178
- multiple access, restriction 17
- multiple access to program libraries 15
- multiple access to type-related libraries 18
- multiple selection 31
  - examples 32
  - symbols 31

**N**

- NAM statement 142
- negative acknowledgment 67
- NEWFORM processing operand 234
- non-delta member 58
- non-delta storage method 58
- NOP statement 144
- NUM statement 145
- numbering in the check field 252
- numbering member records 47, 145, 252

**O**

- object module 22, 28
  - addition 96
  - correction 165
  - correction using UPD 272
- object module library 19, 28
- object module parts, deletion 171
- object modules, correction 54
- OML 75
- operands 83
- operation 82
- organization of delta members 59
- OSM 75
- output format for directories, controlling 249
- output library 12
- output scope, definition 221
- outputting a directory 55, 162

outputting comparison statistics 159, 282  
outputting members 46, 277  
outputting members to files 152  
outputting members to FMS libraries 156  
outputting messages 216  
outputting the statement buffer 201  
overview of statements 85  
OVERWRITE processing operand 235  
overwriting delta members 62  
overwriting members 235

**P**

page turning, control 161  
PAM file 77, 79  
PAM key elimination 75  
PAR statement 147, 203  
phases, correction 183  
physical deletion 47, 61  
    control 215  
PLAM 75  
PLAM error codes 320  
positive acknowledgment 67  
predecessor member 59  
predefining the member type 251  
printing list members 140  
procedure 23, 144  
    call 56  
    storage 56  
processing delta members 266  
processing of members 42  
processing operands 203  
    effect 63  
    overview 205  
    setting 147  
program error 73  
program library 1, 13  
PRT statement 148

**Q**

queries 317

**R**

RANGE processing operand 238  
record format, definition 219  
record ID 48  
record length for macros 20  
record length for member type M 28  
record length for member type R 28  
record length for member type S 28  
record length for object modules 20  
record length for source programs 20  
record numbers 47  
record suppression 244  
records  
    changing 114  
    deleting 113  
    inserting 112  
    replacing 113  
reference conditions, definition 239  
REFERENCE processing operand 239  
remove corrections 175  
renaming delta members 62  
renaming members 55, 142  
renaming symbols 175  
REP record 171  
    conversion 174  
    insertion 176  
replacing records 113  
restricting multiple access 17  
restrictions on sequential libraries 20  
return address 289  
RST statement 150  
run, editor 129  
run mode 70, 150  
    activation 248

**S**

SAM file 79  
SAM/ISAM file 77  
scratch file, EDT/EDOR call 128  
screen overflow control 68, 161  
segment, definition 189, 240  
SEGMENT processing operand 240

- SEL statement 151
  - format 1 152
  - format 2 156
- sequential libraries 20
  - restrictions 20
- sequential library, assignment 41, 137
- setting processing operands 147
- SORT processing operand 242
- sorting a directory 242
- source library 18, 28
- source program 22
  - addition 253
  - assembling 253
  - correction 253
  - correction using COR 269
- statement buffer, output 201
- statement input, control 68, 117
- statements
  - blocked format 84
  - format 82
  - logging 228
  - overview 85
  - syntax 81
- storage mode 15
- storage unit 14
- storing comparison statistics 158
- storing delta members 59
- storing procedures 56
- STRING processing operand 243
- STRIP processing operand 244
- structure of a library 11
- STXIT routine 72
- subroutine interface 289
- SUM processing operand 245
- SUM statement 158
- SUMADD statement 159
- SUMDEL statement 160
- SUMPRT statement 159
- suppressing a record 244
- symbols, renaming 175
- symbols for construction specification 34
- symbols for multiple selection 31
- synchronization 106
- synchronization counter 213

- syntax of statements 81
- SYS statement 160
- SYSDTA 56
- system commands, issuing 160
- system input file 56
- system queries 317

**T**

- table of contents 11
- tape library 20
- TCH statement 161
- temporary library assignment 39
- terminal characteristics, change 161
- TERMINATE processing operand 246
- terminating correction input 116, 188
- terminating correction statements 172
- terminating LMS 133
- terminating test mode 248
- test condition 74
- test mode 70
  - activation 248
  - deactivation 248
  - leaving 150
  - termination 248
- TEST processing operand 248
- text corrections, conversion 174
- text data 24
- text member
  - correction 127
  - creation 127
- text members, correction 53, 108
- text record, correction 169, 186
- TOC processing operand 249
- TOC statement 162
- transferring file attributes 224
- transferring the ISAM key 224
- TXTP record 171
- TYPE processing operand 251
- type-related libraries 18

**U**

- UPD statement 54, 164
  - format 1 165
  - format 2 183
- UPDR statement, old format 181
- use of job switches 74
- USE statement 196
- user exit
  - deactivation 197
  - interface 198
- user interfaces 70
- user program
  - branch 196, 285
  - dynamic loading 197

**V**

- VALUE processing operand 252
- variant number 27