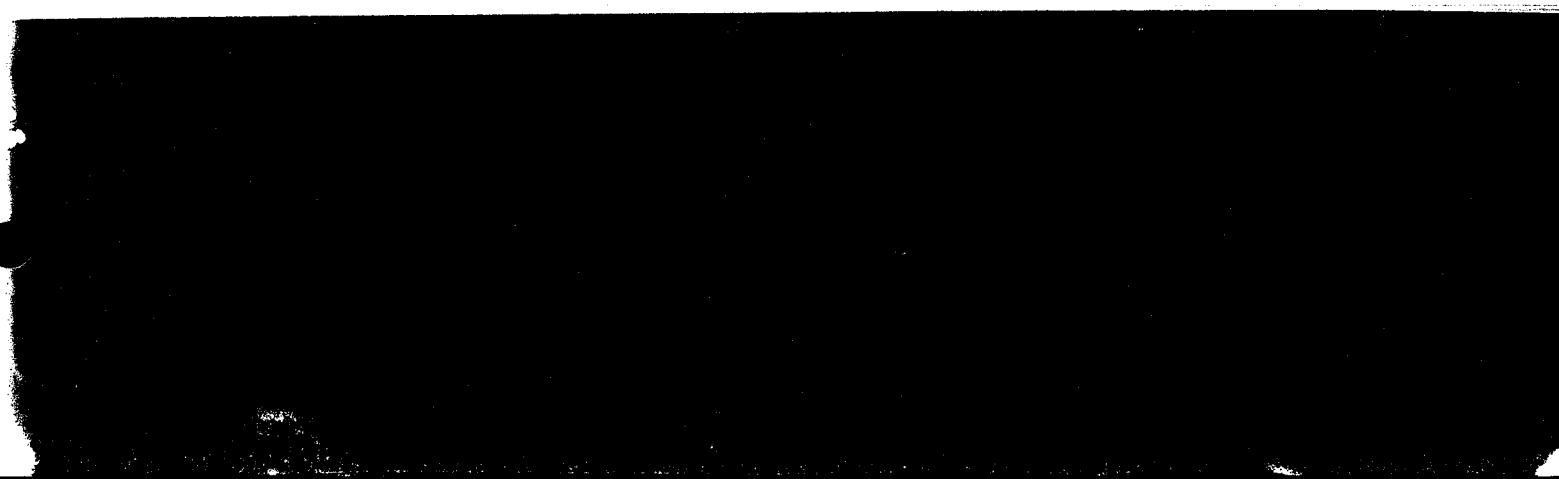
SPERRY+UNIVAC

CDI Library User Interface

Document No. CS-61 Update 1

30 SEP 82



CONTENTS
CONTENTS

1. INTRODUCTION

1.1. SCOPE

This documents the user interfaces using Library Utilities necessary to access or generate library modules or manipulate a library in the CDI environment.

Some of the MIRAM library modules supported for the current release R9.0 are:

- saved run library modules (Type J)
- screen format modules (Type F, FC)
- HELP modules (Type Help)
- MENUs (Type Menu)

The SAT library modules supported for the current release are:

- source
- proc
- object
- load
- group's

2. FUNCTIONS AVAILABLE

The system library interface allows user programs to directly access any system library file. The functions provided are:

- reading a library module
- creating a library module
- deleting a library module
- interrogating a library directory
- updating library module header information

2.1. LIBRARY INTERFACES

2.1.1. Interface Macros

The CDI macros are the only ones used to interface with libraries (thru library utilities). Both imperative and declarative macros are used. The imperatives are:

- OPEN - data management file OPEN
- CLOSE - data management file CLOSE
- DMINP - retrieve a library record
- DMOUT - write a library record
- DMSEL - initialize a library operation
- DMUPD - change module header information

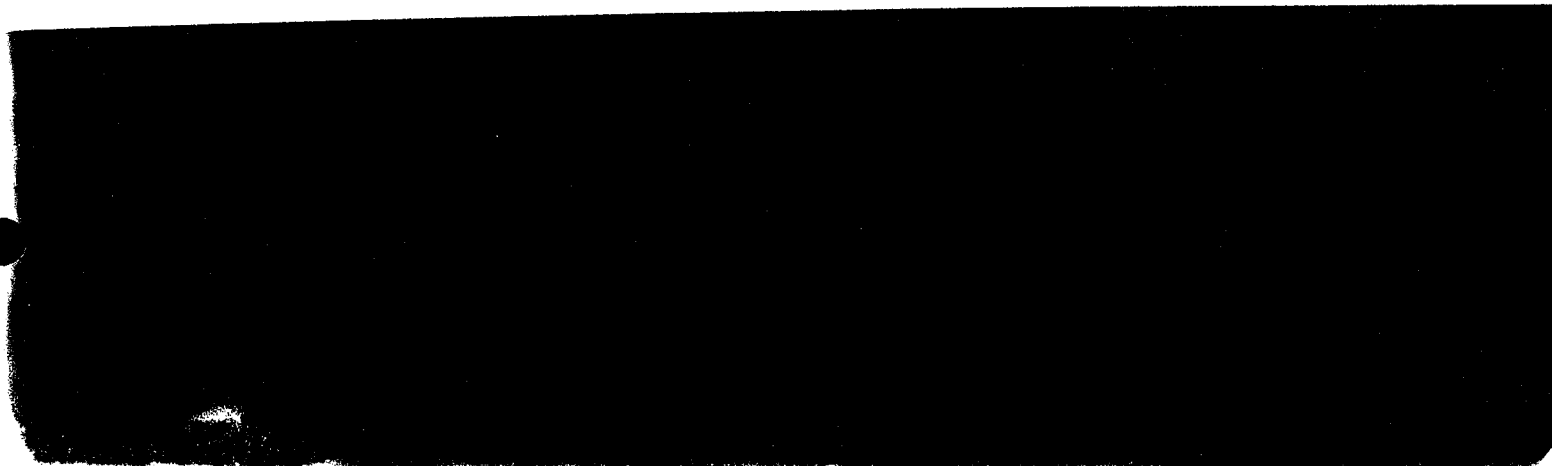
The declaratives are:

- CDIB - library operation control block
- RIB - optionally used only at OPEN

2.1.2. Flow of Control

The CDIB (as defined by data management) is the basic controlling block used by library utilities to communicate with the user program. It is referenced on each imperative macro and will contain status on return.

The user starts by issuing an OPEN imperative to a CDIB which identifies the library file involved (by LFD). This is a standard data management requirement.



The next step depends upon what kind of operation the user wants to do. The operations available are described in Section 2.2 and assume the CDIB is already open. The flow of most operations is:

1. DMSEL - initialize (or identify) operation

2. DMINP/DMOUT/DMUPO - retrieve or present data

3. DMSEL - terminate operation

Step 2 will probably be executed more than once. Step 3 is now always required.

The user must issue a CLOSE imperative macro for each active CDIB before the program terminates. CLOSE *ALL is also acceptable.

2.2. READING AND WRITING LIBRARY MODULES

Library modules are normally read or written one record at a time (using a DMINP or DMOUT imperative). The only exception is when transfer mode (see the DMSEL imperative, section 4.1.2) is used to copy an entire module. Transfer mode reads and writes 256 byte blocks instead of single records.

Library Utilities supports both fixed and variable length records. The record format is specified in the RIB parameter RCFM.

Fixed-record lengths are specified by the RIB parameter RCSZ and may vary from 1-32K bytes. Variable-records have the maximum allowed length specified in the RIB and the actual record length in the first 2 bytes of the 4 byte Record Descriptor Word (RDW) as defined by Data Management (See CS-14, "Common Data Interface (to Data Management)

2.2.1. Reading Library Module Records

1. DMSEL cdibname|(1)|1,LIB,IN,workarea|(0)|0

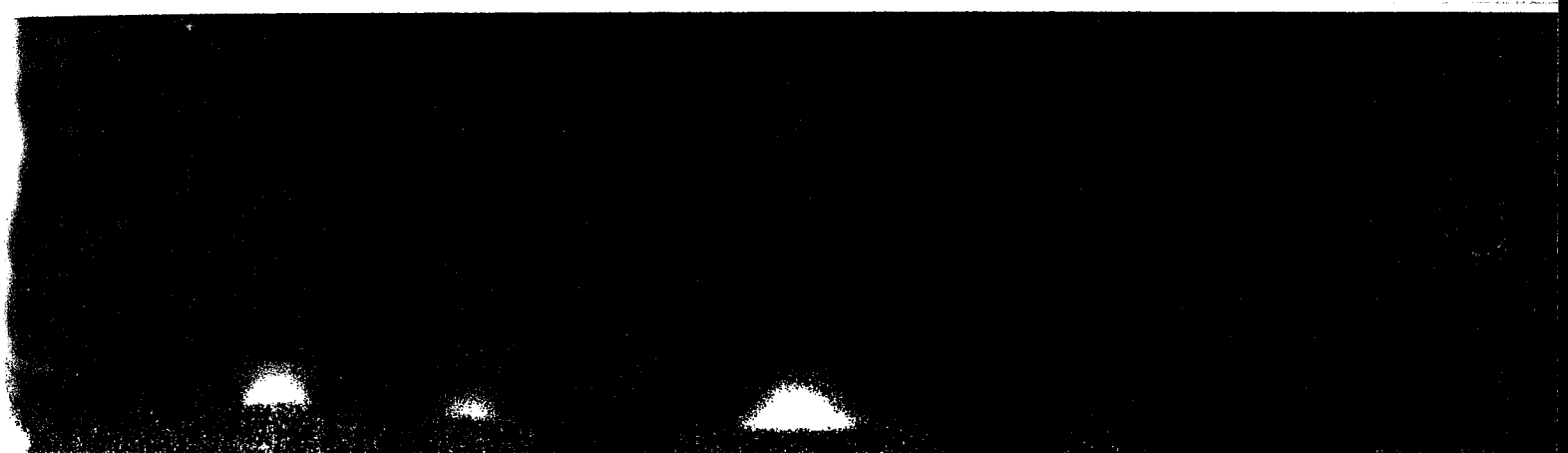
2. DMINP cdibname|(1)|1,workarea|(0)|0

DMSEL initializes the operation and locates the module by the name and type in the workarea (see section 2.4). DMINP moves data records, one at a time, to the user buffer. The user issues as many as needed. Errors and end-of-data are signaled in the appropriate CDIB fields.

2.2.2. Writing Library Module Records

1. DMSEL cdibname|(1)|1,LIB,OUT,workarea|(0)|0

2. DMOUT cdibname|(1)|1, workarea|(0)|0



3. DMSEL cdibname|(1)|1,LIB,ADD

The first DMSEL (out) initializes the operation and specifies the name and type of the module being generated. DMOUT adds records sequentially to the end of the file. The user issues as many as needed. The last DMSEL (ADD) completes the operation by updating the necessary directory indices to include the new module and end-of-library (ENDLIB) marker. If a module exists with the same name and type, it is deleted.

2.3. DELETING A LIBRARY MODULE

DMSEL cdibname|(1)|1,LIB,DEL,workarea|(0)|0

DMSEL initializes the operation and locates the module specified by the name and type in the workarea. The module is marked deleted but not physically removed from the file until it is copied or packed. If no module satisfying the search criteria (full name and type must be specified) an error condition is returned.

2.4. SEARCHING A LIBRARY DIRECTORY

2.4.1. Specific Module Search

The library directory can be searched for a specific module by using a 12 byte key consisting of 8 bytes of name and 4 bytes of type, both padded with blanks as necessary.

Example:

```
|MYMOD   |S   |
|0      7|8 11|
```

2.4.2. Gang Module Search

Library Utilities also permits searches based on partial name and/or partial type. This is called a "gang" search. Partial names and types are padded with binary zeroes.

Example:

```
|MY00000000000|TY0000|
|0            7|8    11|
```

where 00 stands for x'00'. The search is for any module with a name beginning 'my' and a type beginning 'TY'.

If the module name or type does not matter, a x'00' is placed in the first position of the appropriate key field.

Example:

```
|0000000000000000|F   |
|0               7|8  11|
```

where 00 stands for X'00'. The search is for any module whose type is 'F'.

If the entire key is X'00's, Library Utilities will accept any module found in the file.

2.4.3. Performing a Library Directory Search

The required sequence of imperative macros is:

1. DMSEL cdib,LIB,HIN,(0)
2. where R0 points to the 12-byte module key.
3. DMINP cdib,(0)

DMSEL initializes the operation.

DMINP retrieves the header of the first (next) module whose name and type key satisfies the search criteria, and puts it in the workarea specified by R0. This imperative may be repeated as often as desired and will set the end-of-file condition in the CDIB when no more modules satisfying the search criteria are found.

2.5. CHANGING LIBRARY MODULE HEADER INFORMATION

1. DMSEL cdibname|(1)|1,LIB,HIN,workarea|(0)|0
2. DMINP cdibname|(1)|1,workarea|(0)|0
3. Make desired changes to header name, type, or comment fields.
4. DMUPD cdibname|(1)|1,workarea|(0)|0

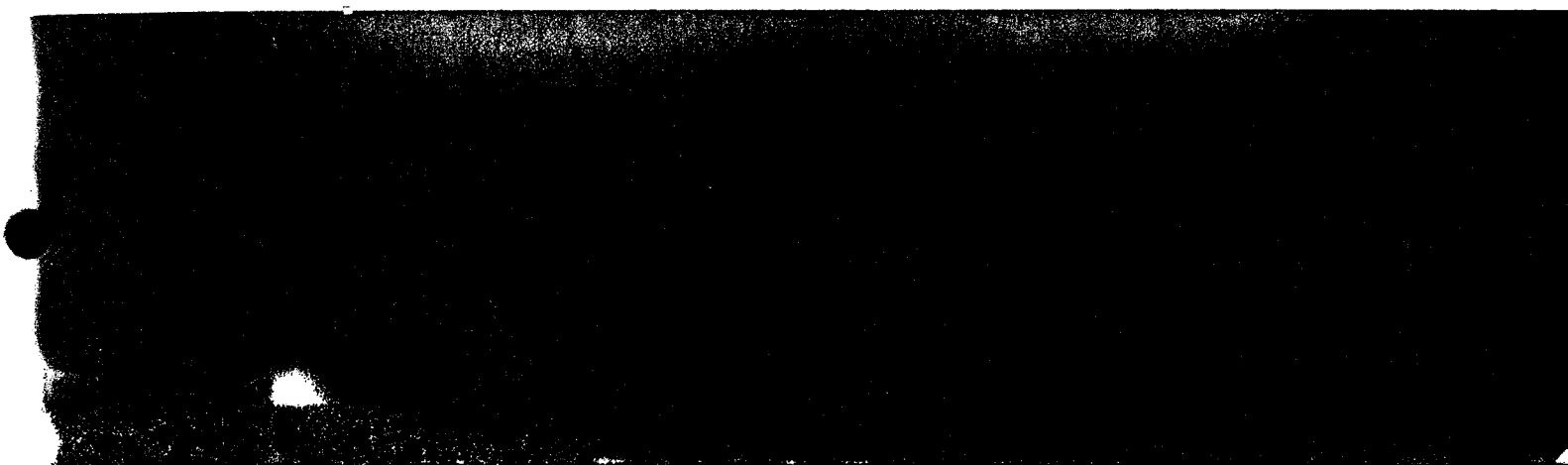
DMSEL initializes the operation. DMINP locates the module and returns its header record; if no module is found, an error is returned. The user alters the name, type, or comment fields as desired and then writes the altered header and updates the directory with a DMUPD command. If the new module name and type matches an existing module name and type, the change operation is suppressed and an error is returned.

3. DESIGN TRADE-OFFS AND PRODUCT OBJECTIVES

The goals in providing library utilities through a CDI interface re are:

1. To provide a standard library interface consistent with other data management interfaces. We can eventually allow users to use library utilities (which we never have had before). The interface is easily learned as it is a derivative of the CDI system standard.

2. To provide device independence for sequential source INPUT/OUTPUT users. Some of the special commands (such as DMSEL module name for input) should be optionally performed automatically as part of OPEN. This would require more information on the job control device allocation set. It would allow users to use the same routine to read/ write from either a sequential device or a library element.



4. INTERFACE DESCRIPTION

4.1. USER INTERFACE

4.1.1. Declarative Macros

4.1.1.1. CDIB Macro

The CDIB is the controlling block for any library operation in progress. Its use and format are defined in CS-14, "Common Data Interface (to Data Management)".

4.1.1.2. RIB Macro

The RIB (Resource Information Block) is used to present Library Utilities with information regarding the file structure and the user's environment. The subset of RIB parameters pertaining to library file access is shown below; a full description of the RIB macro can be found in CS-14 "Common Data Interface (to Data Management)".

Format:

```
label   RIB     [,ACCESS={EXC|SRDO}]
                [,LIBADD={NO|YES}]
                [,LIBINIT={NO|YES}]
                [,MODE={SEQ|RAN}]
                [,NWAIT={NO|YES}]
                [,RCFM={FIX|VAR}]
                [,RCSZ={256|n}] *
                [,RECPASS={NO|YES}] *
                [,SKAD=SYMBOL] *
                [,WKFM={NO|VAR|VARI}]
```

ACCESS — Specifies the type of file sharing permitted:

- EXC — Only 1 access path to the file is permitted at any time. This path has exclusive read/write use of the file. (This is the default).
- SRDO — Multiple access paths to the file are permitted, but only to read. Write operations are not allowed.

LIBADD — Specifies that an automatic DMSEL (ADD) is to be done when the file is CLOSEd.

LIBINIT — Specifies that the file is to be initialized when it i

MODE* Specifies SEQuential or RANdom access to the file.

NWAIT Specifies whether the task should be waited when a file share environment incompatibility occurs. (See the ACCESS para- meter).

RCFM Specifies the record format.

FIX fixed length records

VAR variable length records. The RCSZ parameter gives the maximum allowable record size. The first 4 bytes are the record descriptor word (RDW, see CS-14).

RCSZ* For fixed records, this is the record size in bytes. For variable records, this is the maximum allowable record size.

RECPASS* Specifies if record boundaries are ignored when reading a file. "NO" will cause L.U. to start each input operation at the beginning of a record.

SKAD* The location of a fullword which contains the module relative record number when reading randomly (MODE=RAN). The first record in the module is relative record 1.

WKFM Specifies the workarea format as fixed or variable (4 byte record descriptor word followed by data). The workarea format may be different from the record format.

NO The workarea and record formats are identical. (This is the default).

VAR The workarea format is variable. For output, the user must specify the effective record size in the first halfword of the RDW. On input, L.U. will put the record size in the RDW. The workarea must be large enough to hold the largest record which might be returned.

VARI Same as VAR for output operations. On input, the user requests a number of bytes to be moved in the RDW. If the record is longer than this amount, the record is truncated and the CDIB will reflect this. If the record is too short, L.U. will alter the RDW to hold the correct record length.

*These parameters are for MIRAM library files only.

4.1.2. Imperative Macros

4.1.2.1. DMSEL Macro

```
DMSEL    {cdibname|(1)|1}
         ,LIB
         ,{ADD|DEL|HIN|IN|NEXT|OUT|XIN|XOUT}
         ,{workarea|(0)|0}
```

Positional parameters 1 and 4 are described in CS-14 "Common Data Interface (to Data Management)".

Positional parameter 2 indicates that this directive pertains to a library file and Library Utilities should be activated.

Positional parameter 3 gives the type of action to be done:

ADD create a directory entry for the current module and add it to the file. Positional parameter 4 does not apply.

DEL delete the module whose name and type key appears in the workarea.

HIN initialize the input operation to return only module headers. Search criteria are specified in the workarea (see section 2.4).

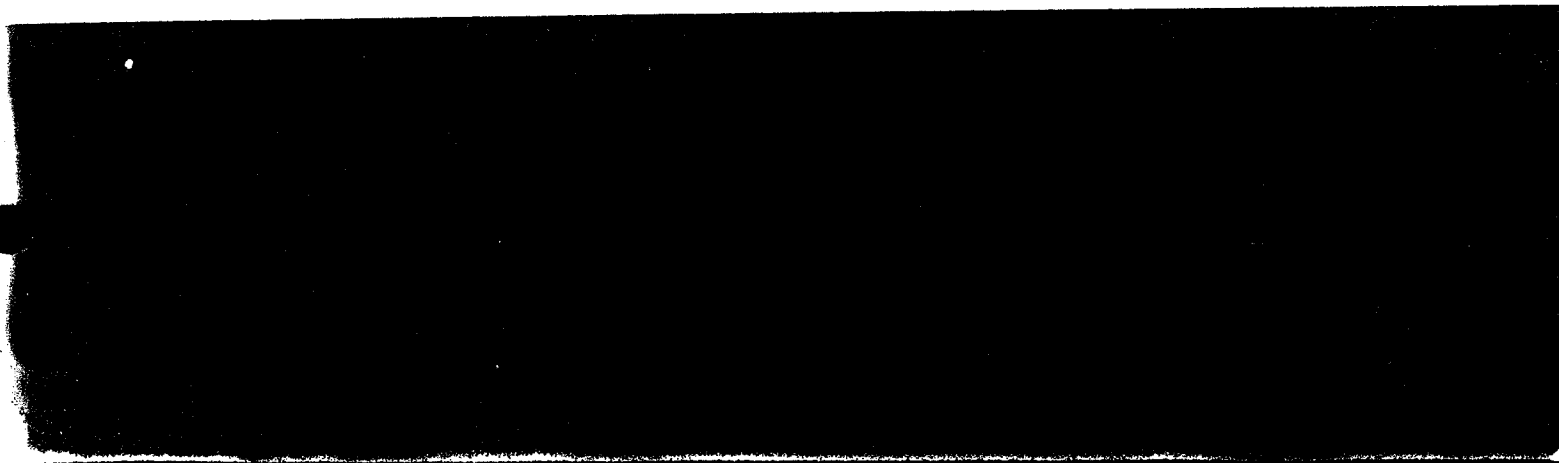
IN initialize the input operation to return module records.

NEXT initialize the input operation for the next module in file with a name and type key satisfying the search criteria previously specified on the DMSEL in macro (see section 2.4).

OUT initialize the output to write module records to the end of the file. The module name and type key is specified in the workarea.

XIN initialize the transfer mode input operation. Each subsequent DMINP will return a 256-byte block from the module beginning with the module header. It must be paired with XOUT for output.

XOUT initialize the transfer mode output operation. Each subsequent DMOUT will write a 256-byte block to the file. L.U. expects the first block to contain the module header. It must be paired with XIN for input.



In each operation where a name and type key is referenced, a 12-byte contiguous area consisting of an 8-byte name and 4 byte type fields is expected. (See section 2.4).

4.1.2.2. DMINP/DMOUT Macros

The DMINP reads and DMOUT writes records (or blocks in the case of transfer mode) to or from the file. Both are described in CS-14 "Common Data Interface (to Data Management)".

4.2. OPERATOR INTERFACES

N/A

4.3. DATA BASES

The proposed structure of the MIRAM library file will be presented in an appendix.

The definition of the SAT library file structure exceeds the scope of this document.

5. ENVIRONMENT CHARACTERISTICS

5.1. HARDWARE REQUIRED

Unknown.

5.2. RESTRICTIONS

The MIRAM libraries (and library utilities) exist only in the CDI environment. MIRAM libraries will be inaccessable in the DTF only system.

5.3. COMPATIBILITY

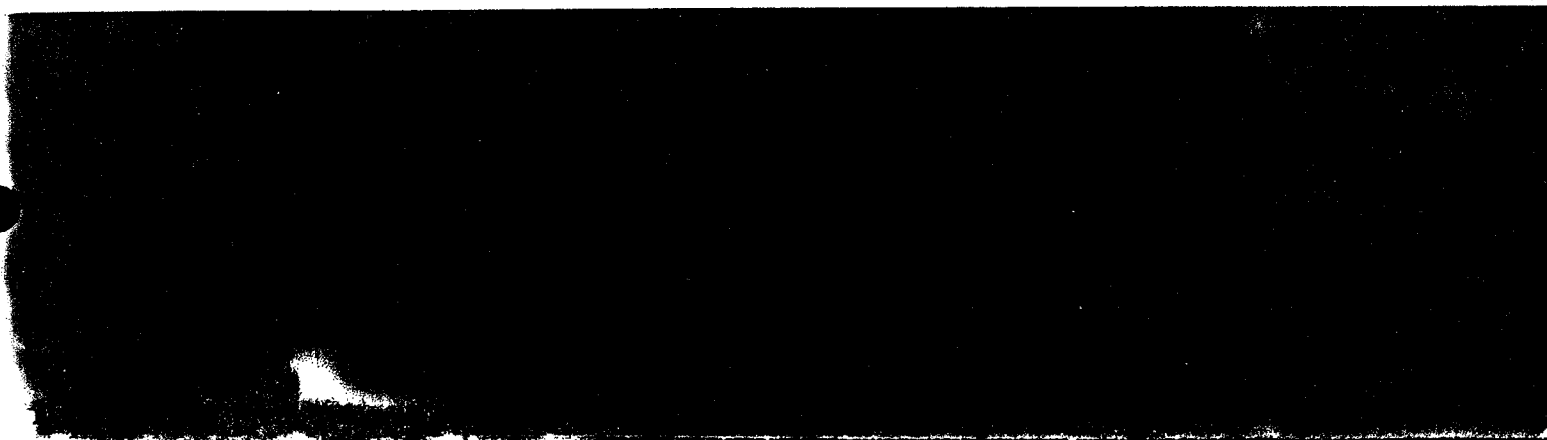
N/A

6. ARM

To be supplied.

7. PERFORMANCE

To be supplied.

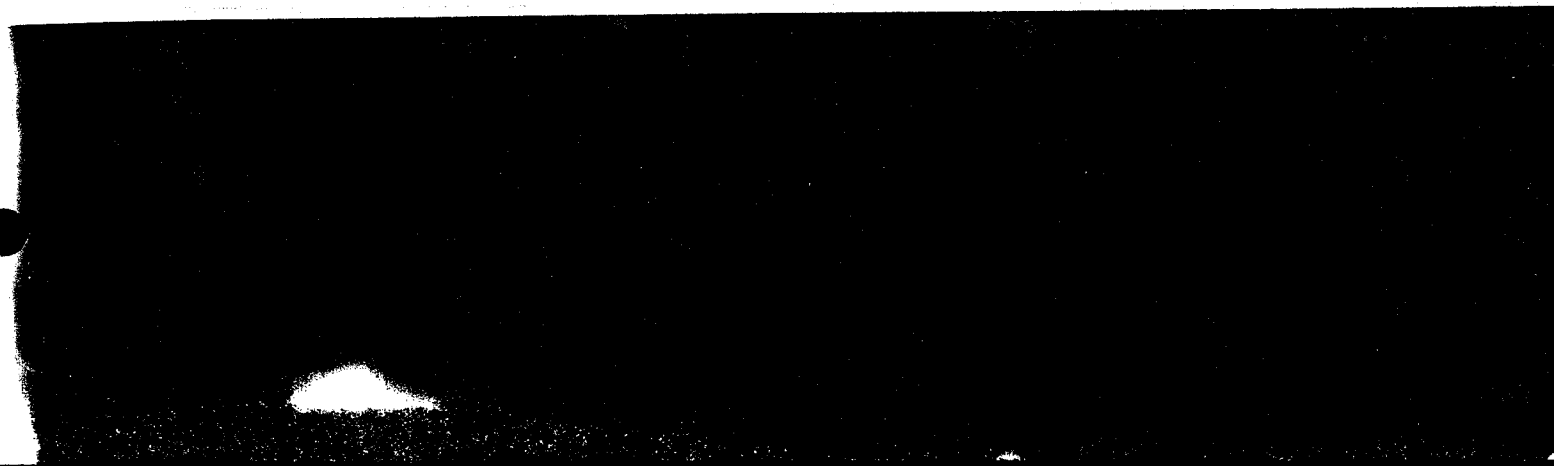


8. STANDARDS

To be supplied.

9. STANDARDS DEVIATIONS

N/A



10. DOCUMENTATION

N/A

11. SUPPORT

N/A

APPENDIX A. PROPOSED STRUCTURE FOR MIRAM LIBRARIES:

This format for MIRAM library files is based on the Nailos-Holt- Snyder letters (10/17/78 - 11/20/78).

A.1. DATA PARTITION

The data partition will consist of 256 byte fixed length non-keyed records. Neither deleted records nor mixed keyed/non-keyed records will be used. It is felt that neither feature is required and their inclusion would require a one byte record control block (RCB) in each record. The RCB would break otherwise contiguous text data (for block modules) and unnecessarily complicate the design.

A module consists of a set of contiguous blocks. The first block is the header block, which occupies one full block and contains the number of blocks consumed by the module. The next blocks contain the module data. All pointers within the header which point to blocks within the module are relative to the module header itself. Copying the module from one file to another can be block for block without modification.

The first module will be written starting in the first block of the data partition, followed by an ENDLIB block. When a second module is created, its header will be written to the same block as the ENDLIB and a new ENDLIB block will be written at the end of the module. There should only be one ENDLIB block in every file. If this is not the case, the file is considered corrupted.

The header block format is the same for all modules types as well as BOG/EOG records and contains these fields:

- Module name
- Module type
- Creation Date
- Creation Time
- Block (module relative) displacement of text block(s)
- Total number of blocks consumed by the text
- Total number of blocks consumed by the entire module
- Flag Field(s)
- Record size (max record size of variable records)
- Reserved fields

- Comment

A.2. DIRECTORY PARTITION

The directory is a set of MIRAM indices maintained by MIRAM and manipulated through index-only operations by library utilities. Indices are maintained lexically ordered. Since none of the data records are keyed, library utilities will be aware of the directory entries necessary for each module and add each using an index-only write.

BOG/EOG records occupy an entire block and have the same format as a header record (roughly) with these exceptions:

- Group Name is the module name
- 'BOG' or 'EOG' is the module name
- Number of module blocks is set to 1

no. CS-61 rev. no. _____ date: 3/14/79

OS/3 COMPONENT PRODUCT SOFTWARE DESCRIPTION

name: CDI Library User Interface

component no.: _____

author: M. B. Todd

hardware systems: ☐ SUL ☐ 90/25 ☐ 90/30 ☐ 90/40

☐ _____

JOHN TOWNSEND
UNIVAC VANCOUVER

MIKE TODD 215-542-4945
E8148

4945

ABSTRACT:

FROM Jerry GABLE — These Works, Specs not Current but should be OK for our purposes Call if you have further questions

USE =LIB

APPROVALS:

section manager:	director
group manager:	
director:	

1.0 Introduction

1.1 Scope

This documents the CDI user interfaces to system libraries using library utilities. This one necessary to access or generate library modules and manipulate the library is contained here.

The MIRAM library modules supported for the current release (R7.0) are: R7.1

- o saved run library modules
- o screen format modules

The SAT library modules supported for the current release are:

- o source
- o proc

NO OBJECT LOAD

2.0 Functions Available

The system library interface allows user programs to access directly any system library file. These functions are provided:

- reading a library module
- creating a library module
- deleting a library module
- interrogating a library directory

2.1 Library Interfaces

2.1.1 Interface Macros

The CDI macros are the only ones used to interface with libraries (thru library utilities). Both imperative and declarative macros are used. The imperatives are:

- OPEN - data management file OPEN
- CLOSE - data management file CLOSE
- DMINP - retrieve a library record
- DMOUT - write a library record
- DMSEL - initialize a library operation (usually)

The declaratives are:

- CDIB - library operation control block
- RIB - used only at OPEN (as per data management)

2.1.2 Flow of Control

The CDIB (as defined by data management) is the basic controlling block used by library utilities. It is used to communicate between the user program and the utilities. It is specified on each imperative macro and will contain status on return.

The user starts by issuing an OPEN imperative to a CDIB which identifies the library file involved (by LFD). This is a standard data management requirement.

The next step depends upon what kind of operation the user wants to do. The operations available are described in Section 2.2 and assume the CDIB is already open. The flow of most operations is:

1. DMSEL - initialize (or identify) operation
2. DMINP/DMOUT - retrieve or present data
3. DMSEL - terminate operation

This is an outline of a typical operation. Step 2 will probably be executed more than once. Step 3 is not always required.

The user must issue a CLOSE imperative macro for each CDIB active before the program terminates.

2.2 Reading and Writing Library Modules

This interface allows the user to read or write library modules a record at a time. Records of a library module are passed between library utilities and the user program as a result of a DMINP or DMOUT macro instruction.

2.2.1 Reading a Library Module

This sequence of imperative macros is required:

1. DMSEL cdib,LU,IN { ~~ELEMENT=name,TYPE=type~~ (0) / NEXT }

2. DMINP cdib, [(0) / workarea]

DMSEL initializes the operation by locating the module and returning the header record in the data buffer (IOAREA1). Two types of operation are permitted:

- o select element of specified name and type
- o select the next sequential element in file.

DMINP moves data records to the user buffer. The user issues as many of these as required. When the modules is exhausted, an end of data condition is set in the CDIB.

How do you determine end of module VS end of file ?

2.2.2 Writing a Library Module

This sequence of macros is required:

1. DMSEL cdib,LU,OUT,~~ELEMENT=name,TYPE=type~~ (0)
2. DMOUT cdib,[ø / workarea]
3. DMSEL cdib,LU,ADD

DMSEL (out) initializes the operation.

DMOUT accepts user data records sequentially and adds them to the end of the module being generated. The user issues as many calls as required.

DMSEL (ADD) completes the operation by adding the module to the file. All required directory indices are created by library utilities on this call. If a module exists with the same name, it is deleted.

When this call is omitted, this is the result:

- the module is not added to the file
- a module with the same name is not deleted
- the integrity of the file is maintained

2.3 Deleting a Library Module

To delete a library, use the DMSEL macro:

```
DMSEL   filename, LU, DEL, (0)
```

2.4 Interrogating a Library Directory

(To be described in a later revision).

3.0 Design Trade-Offs and Product Objectives

The goals in providing library utilities through a CDI interface are:

1. To provide a standard interface consistant with other data management interfaces. We can eventually allow users to use library utilities (which we never have had before). The interface is easily learned as it is a derivative of the CDI system standard.

2. To provide device independence for sequential source INPUT/OUTPUT users. Some of the special commands (such as SELECT module name for input) should be optionally done automatically as part of OPEN. This would require more information on the job control device allocation set. It would allow users to use the same routine to read/write from either a sequential device or a library element.

4.0 Interface Description

4.1 User Interface

4.1.1 Declarative Macros

4.1.1.1 CDIB Macro

The CDIB is the controlling block for any library operation in progress. Its use is consistent with the data management interface. His format is:

```
filename  CDIB
```

The format of the CDIB is defined in CS-14.

4.1.1.2 RIB Macro

NAME	OPERATION	OPERAND
label	RIB	[,ACCESS= EXC / EXCR / ~~SHR~~]
		[,BFSZ= 256 / n]
		[,INDA=symbol] *
		[,IOAl=symbol]
		[,IORG=(r)]
		[,NWAIT= NO / YES]
		[,RCSZ= 256 / n]
		[,WORK= NO / YES]

USE=LIB

— Workaround for NO Option?

* Used for MIRAM library files only.

All parameters are optional. Omitted parameters take on the underlined value.

The INDA and IOAl parameters are not required, either. If they are omitted, the required space will be acquired from dynamic main storage. If they are specified, they are subject to the constraints:

- For SAT library files, INDA is ignored and unused
- For MIRAM library files, either specify both INDA and IOAl, or omit both. When you specify them, the INDA buffer must immediately precede and be contigious to the IOAl buffer in main storage.

Parameters:

ACCESS=option — defines the disk file lock used. (See CS-14 for details).

BFSZ={256 / n} — The disc buffer size (parameter IOA1). It must be a multiple of 256.

INDA=symbol — The symbolic address of the index processing buffer. It must be half-word aligned. Its length is 256 bytes.

IOA1=symbol — The symbolic address of the disk buffer. It must be halfword aligned.

IORG={2 / 2-12 / r} — r is the general register used to point to the current record when the user is not using a workarea. This parameter is ignored if work=yes is specified.

NWAIT={NO / YES} — Specifies whether to return on a file lock error. (See CS-14 for details).

RCSZ={256 / n} — The size of the user's work area (where records are returned). It should be as large as the largest record expected.

WORK={YES / NO} — Specifies whether records are to be placed in the workarea. If NO is specified, records will be pointed to by IORG. (Note: Source-records will only be expanded when YES is specified.

4.1.2 Imperative Macros

4.1.2.1 DMSEL Macro Format

```
DMSEL   cdib,LU,{ADD}  [,ELEMENT={'string'}][,TYPE={'string'}]...
                {DEL}           {TAG     }      {TAG     }
                {IN }
                {NEXT}
                {OUT}
```

ADD - add the current module being created to the file. This causes the required directory indices to be created. (Keywords ELEMENT, TYPE do not apply).

DEL - delete the named module from the file. This eliminates all directory indices for the module and marks the module header as inactive.

IN - initialize the input of the named module. (This call is required before DMINP macro calls are permitted.) The module header record is placed in the buffer area.

NEXT - initialize the input of the next module in the file. (This call is required before DMINP macro calls are permitted.) The module header is placed in the buffer area.

(Note: if a BOG or EOG is the next file item, it will be returned).

OUT - initialize the output of the named module. (This call is required before DMOUT macro calls are permitted.)

a twelve byte field whose address is in Reggie 5

Where a "named module" is required, it is specified ~~with the keywords ELEMENT and TYPE~~.

- ELEMENT - the 8 character library module/group name
- TYPE - the four character library module/group name

They ~~can be specified as~~ are a continuous 12 byte string ~~(the actual name in quotes) or the tag of the area containing them may be specified~~.

What format is the 4 char TYPE in?

4.1.2.2 DM INP/DMOUT Imperative Macros

The DMINP macro makes a record available for processing The DMOUT macro adds a new record to the library module being created. Both macros are generalized imperatives that can be used to request records from all data management processors.

Their format is:

NAME	OPERATION	OPERAND
[label]	{DMINP / DMOUT}	{ (1) / 1 } {, workarea / (Ø) / Ø }

Parameters:

cdib — the symbolic address of the CDIB that corresponds to the similarly named file assigned through job control.

workarea — specifies a user defined work area that will contain the record transferred.

For DMINP, a retrieved record is placed in the workarea if WORK=YES is specified. Otherwise the record remains in the IOA1 buffer and is pointed to by register IORB.

For DMOUT, the user places records to be output in the workarea.

4.2 Operator Interfaces

N/A

4.3 Data Bases

The proposed structure of the MIRAM library file will be presented in an appendix.

The definition of the SAT library file structure exceeds the scope of this document.

5.0 Environment Characteristics

5.1 Hardware Required

Unknown.

5.2 Restrictions

The MIRAM libraries (and library utilities) exist only in the CDI environment. MIRAM libraries will in inaccessable in the DTF only system.

5.3 Compatibility

N/A

6.0 ARM

To be supplied.

7.0 Performance

To be supplied.

8.0 Standards

To be supplied.

9.0 Standards Deviations

N/A

10.0 Documentation

N/A

11.0 Support

N/A

Appendix A Proposed Structure for MIRAM Libraries:

This format for MIRAM library files is based on the Nailos - Holt - Synder letters (10/17/78 - 11/20/78).

Data Partition

The data partition will consist of 256 byte fixed length non-keyed records. Neither deleted records nor mixed keyed/non keyed records will be used. It is felt that neither feature is required and their inclusion would require a one byte record control block (RCB) in each record. The RCB would break otherwise contiguous text data (for block modules) and unnecessarily complicate the design.

A module consists of a set of contiguous blocks. The first block is the header block, which occupies one full block and contains the number of blocks in the module. The next block (or blocks) contains the module data. All pointers within the module which point to blocks within the module are relative to the module itself. Copying the module from one file to another can be block for block without modification.

The first module will be written starting in the first block of the data partition, followed by an ENDLIB block. When a second module is created, its header will be written in to the same block as the ENDLIB and a new one ENDLIB written at the end of the module. There should only be one EDNLIB block in every file.

The header block format is the same for all module types as well as BOG/EOG records and contains these fields:

- Module (or group) name
- Module type
- Number of blocks occupied by this module
- Flags: - status (deleted/active)
  - index item (yes/no)
  - uniqueness (yes/no)
- Number of blocks in module
- Block (module relative) displacement of data/ text blocks.
- Number of text bytes

Directory Partition

The directory is a set of MIRAM indices maintained by MIRAM and manipulated through index-only operations by library utilities. Indices are maintained lexically ordered. Since none of the data records are keyed, library utilities will be aware of the directory entries necessary for each module and add each using an index-only write.

BOG/EOG records occupy an entire block and have the same format as a header record (roughly) with these exceptions:

- Group Name is the module name
- 'BOG' or 'EOG' is the module type
- Number of module blocks is one
- The unique flag is <u>not</u> set (allowing multiple groups of the same name)

O. Townsend

L I B S I O

Library I/O Access routine

DESCRIPTION

This document describes a subroutine which will allow an assembly language or COBOL program to access library files in SIAM format. Source, copylib, proc and macro elements may be read or written using this routine. LIBSIO is written such that the user can not issue functions out of sequence or functions which will render the file inoperable. An abnormal termination routine is included to insure that the file is terminated properly even if the user's program should abort or be cancelled. The routine has a single entry point and communicates with the user program via a 92 byte area. Both assembly language and COBOL use the same interface.

ASSEMBLY LANGUAGE INTERFACE

The user program communicates all requests thru a simple calling sequence. A 92 byte area must be set up according to the diagram shown below. Register 13 must point to an 18 word save area. The calls to LIBSIO are issued using:

```
CALL LIBSIO,(COMMAREA)
```

```
L   15,=A(LIBSIO
LA  1,=A(LIBPTR
BALR 14,15
```

the communications area is set up and modified as explained in table 3. Shown below is a byte map for this area.

```
LIBPTR DC A(COMM
       DS
```

STARTING BYTE	LENGTH	USED FOR
0	80	80 byte record to or from LIBSIO. If file is input this is supplied by LIBSIO. If file is output the user places his output record here.
80	8	Element name to be read or written
82	1	Element type (s,c,p,m) for source copy, proc or macro in EBCDIC
83	1	LIBSIO function code. See table 1 for functions.
84	1	Error code from LIBSIO. See table 2 for error codes.
85	1	File number. MUST BE EBCDIC '1'

```
EXTRN LI
```

```
LA  1,=A(COMMAREA
```

an example of the communication area is:

```
RECORD    DS    CL80
NAME      DC    C'PAYROLL'
ELTTYPE   DC    C'S'
FUNCTION  DC    C'0'
ERRORS    DS    CL1
          DC    C'1'
```

To use LIBSIO in COBOL the user must set up a working storage area definition as shown below. The linkage with COBOL is done by entering linkage and calling LIBSIO:

```
ENTER LINKAGE.
CALL LIBSIO USING LIBSIO-DATA-AREA.
ENTER COBOL.
```

the data area in working storage is set up similar to:

```
01  LIBSIO-DATA-AREA.
    02  RECORD              PIC  X(80).
    02  MODULE-NAME         PIC  X(8) VALUE SPACES.
    02  MODULE-TYPE         PIC  X    VALUE 'S'.
                             This may be s,c,p or m with
                             s = source
                             c = copy
                             p = proc
                             m = macro
    02  FUNCTION CODE       PIC X VALUE '0'.
                             this function code is taken
                             from table 1.
    02  ERROR-CODE          PIC X VALUE '0'.
                             this is returned by LIBSIO
                             it can be found in table 2.
    02  FILLER              PIC X VALUE '1'.
```

Six function are provided by LIBSIO which include: open input, open output (new), open output (update) close and transfer. The transfer operation is used in place of a read or write, the type of open determines which actual function will be used. The user only issues a transfer function. A file may be opened in either of three ways open input, open output (new), or open output (update), any number of records may be transferred, and the file must be closed.

FUNCTION CODE 0 - OPEN INPUT

This is used to open a file and locate the element. It indicates that a user wishes to read the file. The file is opened, and the directory is searched for a module with the same name as that given in the 8 byte element name field. The user supplied 1 byte module type is also used to insure that the correct type module is located possible error codes from this function are: 0,1,2,3,6,7.

FUNCTION CODE 2 - OPEN OUTPUT (NEW)

This function is used to initiate a new element in the file, when the user wishes to issue write commands. The file is opened as output and a module with the name from the 8 byte module-name area is created. The module type (source, copylib, proclib, or macro) is determined by the 1 byte module-type field supplied by the user. If the element already exists, the file is NOT opened, instead an error condtion (4) is posted. If the user wishes to create an element which already exists he should use function code 3 to open the file. Function code 2 is available to allow the user to prevent overwriting of a module which is already in the file.

FUNCTION CODE 3 - OPEN OUTPUT (UPDATE)

This function code works similarly to function 2 except it allows the user to overwrite an element which may exist in the file. Whether the element was contained in the file before the open or not, a new element will be created and the old one, if there was one, will be deleted automatically. If the user wants to know if the element existed before the creation, he should use an open output (new) function

FUNCTION CODE 4 - TRANSFER A RECORD

This function is used to pass 80 character data records between LIBSIO and the user. If the user has opened the file as input, then a transfer function is the same as issueing a read command. His 80 byte area will be loaded by LIBSIO each time he issues a transfer. End of file is indicated by an error code

'9'. If the user has opened the file for output (either mode) then the transfer command has the same effect as issueing a write command. He should load the 80 byte data area with a record prior to each transfer function. Error codes from this function are: 0,5,6,7,9

FUNCTION CODE 5 - CLOSE THE FILE

This function is used to close the file after processing is complete. The file must be closed for both input and output. After the file is closed it may be opened again as either input or output. When a file is closed, no information about it's past use is retained. Possible error codes from this function are: 0,5,6,7.

TABLE 1

FUNCTION CODE	CODE THAT MAY FOLLOW IT	ERRORS	USE
0	4.	0 1 2 3 6 7	OPEN INPUT
2	4	0 1 2 4 6 7	OPEN OUTPUT(NEW)
3	4	0 1 2 6 7	OPEN OUTPUT(UPDT)
4	4 5	0 5 6 7 9	TRANSFER
5	0 2 3	0 5 6 7	CLOSE

TABLE 2

ERROR CODE	FILE CLOSED	MEANING
0	NO	No error has ocurred - sucessful
1	NO	Open attempted on an open file
2	YES	File could not be opened (not there)
3	YES	Element to be read could not be found
4	YES	Module to be created exists (fcn code 2)
5	YES	Transfer (read/write) or close of a file which is already closed
6	YES	Unrecoverable i/o error
7	YES	Error from proc module- attempt to add a name entry which exists for another proc element.
8	~~NO~~ YES	Function code not recognized or file type not set to 's' 'c' 'p' or 'm'.
9	NO	End of file on input file. No data is returned with this condition.

NOTE: Error code 0 is not an actual error, it indicates sucessful completion of the function.

TABLE 3

FIELD	LENGTH	SUPPLIED BY	USE
RECORD	80	LIBSIO	Work area for a read
		USER	Work area for a write
NAME	8	USER	Name of element to be read or written.
TYPE	1	USER	Type of element.
FUNCTION	1	USER	Function which user wants performed.
ERROR	1	LIBSIO	Error or status from libsio
FILE #	1	USER	Must be set to '1'.

LINKING PROCEDURE

To link LIBSIO the user need only have an INCLUDE statement for the subroutine in his linker job stream. A sample of this job stream is shown:

```
// EXEC LNK
// PARAM LIN=RESVS/USERLIB
/$
      LOADM      program
      INCLUDE    program,*
      INCLUDE    LIBSIO,USERLIB
/*
```

JOB CONTROL TO EXECUTE LIBSIO

When executing a program which uses LIBSIO, the SIAM library that is to be accessed must be defined in the job control stream. This usually requires only one statement. The keyword RDWD=YES must be used if the file is ever opened as output. The library file descriptor (LFD) is LIBRARY.

```
// CALCAT LIBS.SRCE.136,FLNM=LIBRARY,USE=RDWD
// CALCAT SYS#.LBS.S18,FLNM=LIBRARY
```

LFD name