Honeywell

LEVEL 64

GCOS

LIBRARY MAINTENANCE

SERIES 60 (LEVEL 64)

LIBRARY MAINTENANCE REFERENCE MANUAL ADDENDUM B

SUBJECT

This Addendum Provides an Index

SPECIAL INSTRUCTIONS

This is the second addendum to AQ28, Revision 1, dated September 1978. Insert the attached pages into the manual according to the collating instructions provided. This addendum corrects the pagination problem which occurred in the previous index, Addendum A.

Note:

Insert this sheet after the manual cover to indicate that the manual has been updated with Addendum B.

SOFTWARE SUPPORTED

Level 64 GCOS Release 0400

ORDER NUMBER

AQ28-01B

March 1980

27133
1480
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

Remove	Insert
i-01 through i-08	i-01 through i-07, blank

File No.: 1N13

4/80
AQ28-01B

LEVEL 64 DOCUMENT LIST

Order Number	Title
AQ02	*Series 100 Program Mode Operator Guide*
AQ03	*Series 100 Conversion Guide*
AQ04	*Series 200/2000 Conversion Guide*
AQ05	*System 360/370 Conversion Guide*
AQ09	*System Management Guide*
AQ10	*Job Control Language (JCL) Reference Manual*
AQ11	*Job Control Language (JCL) User Guide*
AQ13	*System Operation Operator Guide*
AQ14	*System Operation Console Messages*
AQ18	*Operator Reference Manual*
AQ20	*Data Management Utilities Manual*
AQ21	*Series 200/2000 Program Mode User Guide*
AQ22	*Series 200/2000 Program Mode Operator Guide*
AQ26	*Series 100 File Translator*
AQ27	*Series 200/2000 File Translator*
AQ28	*Library Maintenance Manual*
AQ40	*System 3 Conversion Guide*
AQ49	*Network Control Terminal Operation Manual*
AQ50	*Terminal Operations Manual*
AQ52	*Program Checkout Facility Manual*
AQ53	*Communications Processing Facility Manual*
AQ55	*TDS/64 Standard Processor Site Manual*
AQ56	*TDS/64 User Guide*
AQ57	*Standard Processor Programmer Reference Manual*
AQ59	*Unit Record Devices User Guide*
AQ63	*COBOL User Guide*
AQ60	*Interactive Operation Facility*
AQ64	*COBOL Language Reference Manual*
AQ65	*FORTRAN Language Reference Manual*
AQ66	*FORTRAN User Guide*
AQ67	*FORTRAN Mathematical Library*
AQ68	*RPG Language Reference Manual*
AQ69	*RPG User Guide*
AQ72	*Series 200/2000 COBOL to Level 64 COBOL Translator*
AQ73	*IBM COBOL Translator*
AQ82	*BFAS User Guide*
AQ83	*HFAS User Guide*
AQ84	*UFAS User Guide*
AQ85	*Sort/Merge Manual*
AQ86	*Catalog Management Manual*
AQ87	*Library Maintenance User Guide*
AQ88	*I-D-S/II User Guide, Volume 1*
AQ89	*I-D-S/II User Guide, Volume 2*
AQ90	*COBOL Reference Card*
AQ92	*Operator's Reference Card*
AQ93	*RPG Reference Card*
AQ94	*FORTRAN Reference Card*

CONTENTS

TABLES

SECTION I

SCOPE AND PURPOSE

LIBRARY MAINTENANCE (LIBMAINT) is a standard GCOS-64 processor which performs functions and services convenient for the efficient management of libraries. Functions provided allow updating, copying, moving, listing and miscellaneous handling of library members.

The contents of a library can be source language units (including JCL), compile units, load modules, shareable modules or master and sort tables. Libraries are homogeneous in that they contain entities which are all of the same type. A suitable protection mechanism (see Section IV) enforces this important rule. Therefore, in a given LIBMAINT session, only one type of entity can be handled. This type is indicated as a parameter to the processor at JCL level. One will thus indicate : SL, CU, LM, or SM depending on the types of entities to be processed.

LIBMAINT accepts as its input libraries, specified members of a library or sequential files or subfiles defined as separate entities. This input can be stored on disk, tape or card.

The output of LIBMAINT can be one or several members of a library or a sequential file or subfile. The contents can be printed, listed, modified or punched in the same single session.

Actions requested of the LIBMAINT processor are expressed in terms of commands. LIBMAINT commands build up into a powerful language. A command is a directive to LIBMAINT to execute a function such as copy a member from one library to another one or to update the contents of one or several members in a library. A command is built up from the command word and specific parameters, if needed. A small number of commands require additional detailed information on the action to be performed. These actions are known as requests and immediately follow the command in the command input stream. The command stream will generally be defined as being a user supplied input enclosure on cards; the stream can also be a sequential file or a member of a source language library. These latter facilities are of particular help in the case of repetitive operations.

SECTION II

OBJECTS HANDLED

Objects handled by LIBMAINT may be stored in libraries, sequential files or on cards.

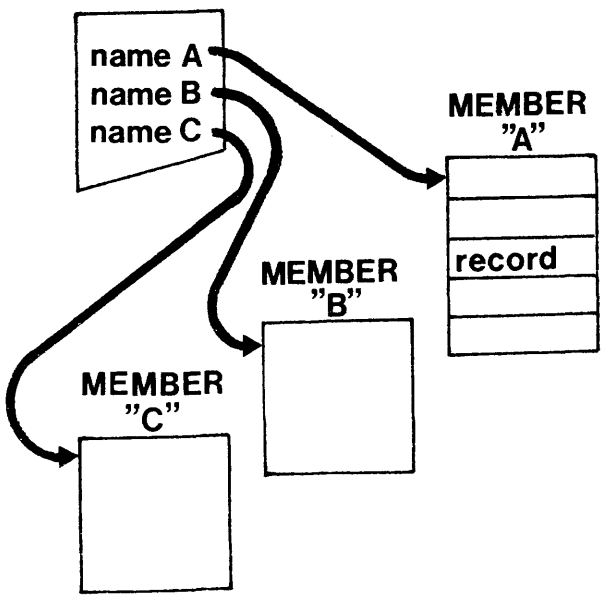
LIBRARIES

Libraries are files with Queued Linked Organization. A library can be viewed as being constituted of:

- a dictionary known as the "directory"
- a number of entities whose names appear in the directory. These are known as "subfiles", "units" or "members".

LIBRARY

DIRECTORY

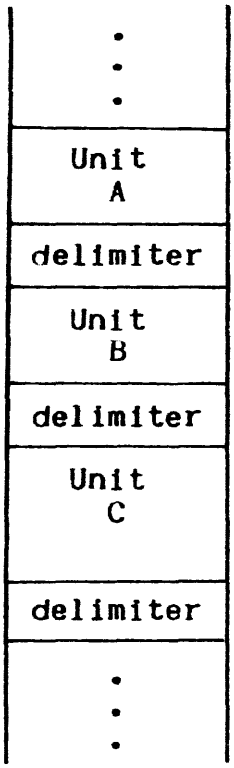


Each subfile is composed of logical records. The format of the logical records and their contents is dependent upon the type of the stored entity. In the case of a source language unit, for example, a record will contain one source line.

Due to their organization, which implies direct access to data blocks, libraries must reside on direct access devices and media, i.e. diskpacks.

SEQUENTIAL FILES

Another method for storing an object acceptable to LIBMAINT is to store all units one after the other on a sequential file. Units need to be separated by some kind of delimiter. This delimiter is dependent upon the type of the stored entities and recognized by LIBMAINT in the proper context.



The sequential file might be any sequential organized media supported by GCOS-64 . i.e. disk, tape, paper tape, etc....

LIBMAINT allows the transfer of units from a sequential file to a library and vice-versa.

CARDS

A unit might also be stored on cards. Cards might be used as input, in which case they are added to the output library or file. These cards are inserted in the command input stream right after the corresponding transfer command or request. If the command input stream is not on card, the cards are replaced by their card images which might have been created and/or modified by the convenient processor.

Cards might also be used as output. This is achieved by the PUNCH command that produces card image(s) of the specified object(s). These card images are generally acceptable as input to LIBMAINT.

The format of the card deck is dependent upon the types of the stored entities. It is checked for validity by LIBMAINT in input mode and produced by LIBMAINT if punched.

SECTION III

UNIT IDENTIFICATION

Unambiguous identification of a unit is vital for the rational management of libraries. In a library, each unit is uniquely identified by:

- the name of the library where the unit is stored
- the name of the unit
- the date and time it was first created
- the date and time it was last modified
- additional information depending on the type of the stored unit.

This unique identification is displayed each time the unit is used. It is updated each time the contents of the unit are changed. A copy of a unit has the same identification (except name and name of containing library) as the original unit if no change has been made in the copy.

The identification of a unit is recorded as a part of the unit, generally as its first logical record.

Raw (SARF) format (see section VIII) source language units have no identification . However, this format of data is seldom used and should be avoided in libraries whenever possible.

EXAMPLE OF IDENTIFICATION DISPLAY:

```
MYUNIT              FROM: JOHN-X.CULIB
CD = 09/26/77 CT = 23:14 MD = 10/07/77 MT = 14:25 SL = FOR CV = 1.0
```

The first line contains:

- The unit's name
- The name of the containing library

The second line always contains:

- CD Creation date (MM/DD/YY)
- CT Creation time (HH : MM)
- MD Modification date (MM/DD/YY)
- MT Modification time (HH : MM)

and possibly:

- SL Source language (for CU and SL)
- CV Compiler version (for CU)
- LK Linker version (for LM)
- MN Modification number (for SL, SM and LM)

SECTION IV

PROTECTION

In order to ensure the security of LIBRARY MAINTENANCE, specific protection measures are taken. These are of two types:

- at library level to ensure that only valid entities are stored in order to be able to later process the library correctly.
- at unit level to ensure that no external action or failure will cause loss of information.

LIBRARY LEVEL PROTECTION

Two types of control are enforced:

Type Protection

It is checked that the library has been allocated with the correct attribute i.e.:

- input library must be of the type specified by the LIBMAINT invocation
- output library must be of the type specified by the LIBMAINT invocation.

In all cases, a library with no type will be accepted as input or output by LIBMAINT. This might however result in errors which will be accepted as input or output by LIBMAINT, which in turn might result in errors which will be often irrecoverable. It is therefore recommended to allocate libraries specifying a correct type.

Characteristics checking

In addition to the preceding verification, all output files or libraries are checked for valid file organization, record format and record size according to the following table 4-1.

TABLE 4-1. OUTPUT FILES and LIBRARY CHECKING

OUTPUT	FILEORG	LIBMAINT TYPE	RECFORM	RECSIZE
Output library LIB	Queued Linked	SL	VB, F, FB } for SARF*	no constraint
		CU	VB	≥ 1024
		LM	F FB	= 1024
		SM	F or FB	= 1024
Output Sequential file OUTFILE	Sequential or Queued Linked if SUBFILE specified in JCL ASSIGN	SL	VB, F, FB } for SARF*	no constraint
		CU	U	≥ 3960
		LM	U	≥ 3960
		SM	F or FB	= 1024

(*) SARF format will be discussed in section VIII.

UNIT LEVEL PROTECTION

In order to be protected against external unforseen actions such as killing the Job or System Crash, LIBMAINT never overwrites an existing unit. Therefore, if LIBMAINT is not allowed by such external events to terminate its writing activity properly, the former version of the object unit still exists in the library. It proceeds as follows:

- creates a temporary unit whose name is "LBMNC" concatenated with the name of the unit.
- deletes the former version of the unit.
- renames the temporary unit with the former version's name.

Even in the highly improbable case of an external event in the latter two steps above, the user will be able to recover the unit under its temporary name.

As a consequence of this policy, enough room must be given in the object library in order to accommodate both the old and new copies of a unit. A good rule is to provide enough space for all units plus the space needed for the largest of these units.

SECTION V

AVAILABLE FUNCTIONS

The aim of this chapter is to give a synoptical view of the LIBRARY MAINTENANCE FUNCTIONS. Two series of tables are given:

- A table stating the available commands in alphabetical order with the specific parameters and/or keywords applying to each type of entity (SL, CU, LM, SM)

- A set of tables giving, for each type of entity, the allowed combinations of inputs and outputs. This second series of tables is preceded by a general overview of LIBMAINT inputs and outputs.

TABLE 5-1. SUMMARY OF COMMANDS

COMMAND	MEANS	SL	CU	LM	SM
CODE	To encode a text	KEY NEW FROM TO REPLACE			
COMM	To comment the report				
COMPARE	To compare two units	LIMIT FROM TO			
CRLIST	To create a unit containing member names.	FROM TO NUMBER REPLACE			
DECODE	To decode a text	KEY NEW FROM TO REPLACE			
DELETE	To delete unit(s) from LIB	FROM TO	FROM TO	FROM TO	SM
EDIT	To invoke the TEXT EDITOR	see requests sect.XIII			
global EDIT	To repeatedly invoke the TEXT EDITOR	NEW FROM TO see requests sect.XIII			
EJECT	To skip to the top of page on the report				
EXEC	To execute a sequence of LIBMAINT commands	VALUES	VALUES	VALUES	

TABLE 5-1 (CONT). SUMMARY OF COMMANDS

COMMAND	MEANS	SL	CU	LM	SM
INDENT	To improve the readability of a GPL unit	NEW FROM TO CM IN LM REPLACE			
INIT	To initialize a SM				SM STN ESSTE REPLACE
LIST	To list entries of directory	FROM TO SIZE	FROM TO SIZE	FROM TO SIZE	SM LKU DIR INFILE DETAILED
LOAD	To load a SM				SM DEBUG REPLACE
LOWER	To convert a source unit into lower case letters	NEW FROM TO REPLACE			
MOVE	To transfer units (*)May not be used with COMFILE	NEW(*) FROM TO TYPE NUMBER END FORMAT INFORM OUTFORM CONTCHAR ENDCHAR REPLACE CHECK	FROM TO END REPLACE	NEW(*) FROM TO END REPLACE	SM REPLACE
PRINT	To print the contents of unit(s)	FROM TO NTAB COPIES			

TABLE 5-1 (CONT). SUMMARY OF COMMANDS

COMMAND	MEANS	SL	CU	LM	SM
PUNCH	To punch unit(s)	FROM TO TYPE ENDCHAR	FROM TO	FROM TO	
RENAME	To rename unit(s)	NEW FROM TO		NEW FROM TO	
RENUM-BER	To renumber unit(s)	FROM TO NUMBER			
SORT	To sort lines of a source unit	NEW FROM TO ASC DESC REPLACE SORTKEY NUMBER			
STATUS	To continue or suspend the processing of commands	EVEN ONLY RESET	EVEN ONLY RESET	EVEN ONLY RESET	EVEN ONLY RESET
TITLE	To introduce a private title in the report				
UNLOAD	Clear the virtual memory				SM SMLIB
UPDATE	To update a unit using requests	NEW FROM TO TYPE NUMBER END FORMAT CONTCHAR ENDCHAR REPLACE			
UPPER	To convert a source unit into upper case letters	NEW FROM TO REPLACE			

LIBMAINT INPUTS AND OUTPUTS

LIBRARY MAINTENANCE supports up to three input libraries, an object library which can be treated both as input and output, an input file, an output file plus miscellaneous inputs and outputs. Selection of units among the input libraries or files will be discussed in section VI. The following diagram illustrates the various LIBMAINT inputs and outputs together with the abbreviations used to denote them.

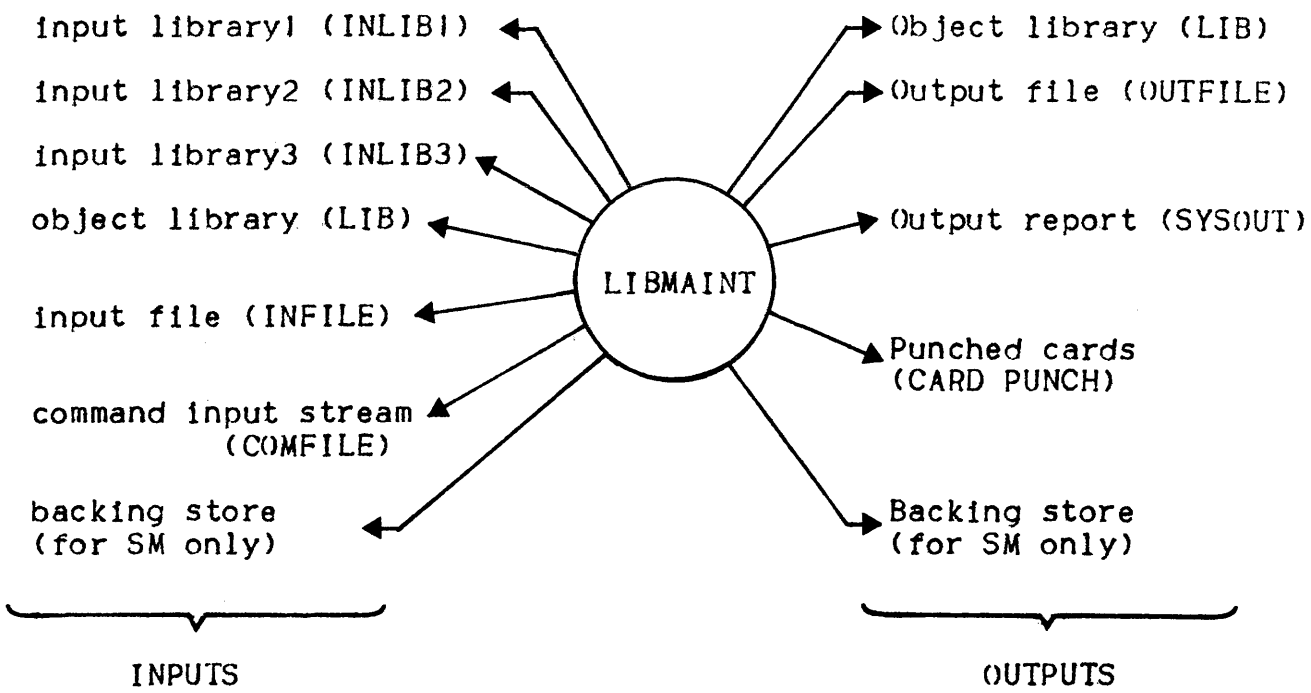


TABLE 5-2. AVAILABLE FUNCTIONS FOR SL

Output / Input	LIB ssf	LIB sarf	OUTFILE ssf	OUTFILE sarf	SYSOUT	CARD PUNCH
LIB ssf	CODE (*) COMPARE (*) CRLIST (*) DECODE (*) DELETE (*) EDIT (*) EXEC INDENT (*) LOWER (*) MOVE (*) RENAME (*) RENUMBER (*) SORT (*) UPDATE (*) UPPER (*)	MOVE (*)			LIST (*) PRINT (*)	PUNCH (*)
INLIB{1,2,3} ssf	CODE (*) COMPARE (*) CRLIST (*) DECODE (*) EDIT (*) EXEC INDENT (*) LOWER (*) MOVE (*) SORT (*) UPDATE (*) UPPER (*)	MOVE (*)	MOVE (*)	MOVE (*)	LIST (*) PRINT (*)	PUNCH (*)
INLIB{1,2,3} sarf	MOVE (*)	MOVE (*)	MOVE (*)	MOVE (*)	PRINT (*)	
LIB sarf	MOVE (*)	MOVE (*) RENAME (*)	MOVE (*)	MOVE (*)	PRINT (*)	
Infile ssf	MOVE (*)	MOVE (*)	MOVE (*)	MOVE (*)	LIST (*) PRINT (*)	PUNCH (*)
Infile sarf	MOVE	MOVE	MOVE	MOVE		
COMFILE ssf/sarf	MOVE	MOVE	MOVE	MOVE		

(*) : More than one unit can be processed in a single command (star convention; etc. See section VI).

ssf/sarf : Distinction between these two formats will be discussed in section VIII.

TABLE 5-3. AVAILABLE FUNCTIONS FOR CU

output / input	LIB	OUTFILE	SYSOUT	CARD PUNCH
LIB	MOVE (*) DELETE(*)		LIST (*)	PUNCH (*)
INLIB 1,2,3	MOVE (*)	MOVE(*)	LIST (*)	PUNCH (*)
INFILE	MOVE (*)		LIST (*)	
COMFILE	MOVE (*)			

(*) More than one unit can be processed in a single command (star convention, etc... see section VI)

TABLE 5-4. AVAILABLE FUNCTIONS FOR LM

input \ output	LIB	OUTFILE	SYSOUT	CARD PUNCH
LIB	MOVE (*) DELETE(*) RENAME(*)	LIST (*)		PUNCH (*)
INLIB 1 2 3	MOVE (*)	MOVE (*)	LIST (*)	PUNCH (*)
INFILE	MOVE (*)		LIST (*)	
COMFILE	MOVE (*)			

TABLE 5-5. AVAILABLE FUNCTIONS FOR SM

input \ output	LIB	BACKING STORE	SYSOUT	OUTFILE
LIB	DELETE UNIT	LOAD	LIST	
INLIB1	MOVE			MOVE
BACKING STORE		UNLOAD		
INFILE	MOVE		LIST	MOVE

(*) More than one unit can be processed in a single command (star convention, etc ... see section VI)

SECTION VI

BASIC LANGUAGE STRUCTURE

The LIBMAINT command input stream is made up of a series of commands, requests and literal text. Requests and literal text are subordinated to a command which is the highest level structure in the LIBMAINT language. The purpose of this chapter is to underline the characteristics which are common to all LIBMAINT commands together with the basic structure of the language.

COMMANDS

A command has the following general format:

<VERB> <POSITIONALS>, <PARAMETERS>;

- <VERB> is the name of the command
- <POSITIONALS> stands for a sequence of positional parameters separated by commas or spaces.
- <PARAMETERS> stands for a sequence of keywords or parameter values. A parameter value is a parameter keyword followed by the equal sign (=) and by a value assigned to this parameter.

All positional parameters, keyword and parameter values are separated by commas or spaces(commas or spaces or a combination of both may be used as delimiters between parameters and positionals). One or several spaces may appear inside the body of the command except within the VERB, keywords or values. The command terminates with a semi-colon. A command may span several cards or records but no more than one command may appear on a single card or record.

Example:

```
,MOVE,    ,INLIB1:UNIT1,,    ,REPLACE, NUMBER = (10,10);,
```

verb positional parameters

Positionals, parameters and values other than library keywords and member-names are described under each command description in the following chapters. This chapter will concentrate on the rules for naming members and retrieving them.

As explained earlier LIBMAINT may take its input from any of the following: INLIB1, INLIB2, INLIB3, INFILE, LIB, COMFILE.

The association of the above keywords with an actual library or file is made by JCL statements (see USER GUIDE). Libraries or files are to be specified only for input, as output is always LIB or OUTFILE depending on the JCL used for activating LIBMAINT.

Search rules are the rules that govern the choice of the input library in which a unit is to be selected. There are two possible cases:

- The library is explicitly stated as a qualifier to the name of the member. This is achieved by means of the library or file keyword prefixed to the name by a colon.

  Example:
  MOVE INLIB1 : UNIT1 ;
  MOVE INFILE : UNIT2 ;

- The library is not explicitly stated in which case the following rules apply :

  - if command is DELETE, RENAME or RENUMBER, the assumed library is LIB.

  - if more than one member is processed in a command by means of a list of member names, the assumed input library is LIB.

  - in all other cases, the named member is searched for, first in INLIB1 then in INLIB2, INLIB3 and last in LIB until a unit with the specified name is found.

NOTE: LIBMAINT SM always works in LIB except for command MOVE which operates on INLIB1 or INFILE. INLIB1 is never indicated. INFILE needs to be specified when required.

MEMBER NAME

A member name is composed of a string of alphabetic (A-Z), numeric (0-9) and special characters. The allowed special characters are the underscore (_), the minus (-) and period (.) symbols. There is no restriction on the first character and the maximum length of a name is 31 characters (30 characters for a SM or LKU).

IMPORTANT NOTE:

THE FOLLOWING DOES NOT APPLY TO LIBMAINT SM.

As stated above, a member name may be prefixed by any of the following:

COMFILE :
INLIB1 :
INLIB2 :
INLIB3 :
LIB :
INFILE :

The library keyword is separated from the member name by a colon (:). Spaces are allowed before and after the colon.

IMPORTANT NOTE :

THE FOLLOWING DOES NOT APPLY TO LIBMAINT SM

LIST OF MEMBER NAMES

Most LIBMAINT commands can operate on more than one member. This is stated by indicating a list of members in place of the member name in the command. There are three basic means through which a list of members can be stated :

- by an explicit list
- by an indirect list
- by the "star convention"

Explicit List

An explicit list of member names is formed by a list of names separated by commas or spaces and enclosed between parentheses. All members must belong to the same library; if none is specified, LIB is assumed. If member-names have a common prefix or suffix, these may be written outside the parentheses. These will act like factors in classical arithmetic. No space is allowed between the prefix and the left parenthesis or between the right parenthesis and the suffix.

Examples :

- (LIBEANAL, LIBERPRINT, LIBEREAD)
  or
  LIBE(ANAL, PRINT, READ)

- (LIBSLERR, LIBCUERR, LIBLMERR)
  or
  LIB(SL, CU, LM)ERR

If an explicit list of member names is prefixed with symbol not (¬), this means that all members except those indicated in the list are to be processed.

Examples :

¬(LIBA, LIBB, LIBC)

or

¬LIB(A, B, C)

-¬(XXABYY, XXCYY, XXDEFYY)

or

¬XX(AB, C, DEF) YY

IMPORTANT NOTE :

THE FOLLOWING APPLIES ONLY TO LIBMAINT SL

Indirect List

A list of the members to be processed may be given in a unit of a source language library. This list might have been created, for example, by the CRLIST command (see section VIII).

To denote that the members list is given in a unit, one specifies the unit name, possibly qualified by a library keyword, enclosed between " < " and " > " (search rules apply there).

Examples :

- MOVE INLIB1 : < LIB : SFLIST1 > ;

The members of INLIB1 whose names are contained in unit SFLIST1 of LIB will be moved to LIB

- PRINT INLIB1 : < LIST2 >

- LIST < LIST3 >;

The "not" convention may also be used with an indirect name list. This means that all members, except those whose names are listed in the unit, are to be processed.

Examples :

- PRINT INLIB1 :¬< INLIB2 : SFLIST4 >

- DELETE ¬< LIST5 >

IMPORTANT NOTE:

THE FOLLOWING DOES NOT APPLY TO LIBMAINT SM

Star Convention

The star convention is a device which allows the specification of a name list in terms of a given pattern. All names that match the pattern are selected for processing. Constant sections of the pattern are expressed as such, variable sections are denoted by an asterisk or a star (*). When a name is found to match the pattern, the "*" replaces 0 to n characters.

Examples :

Pattern	Matches	Does not match
A*B	AB AXYYXB AXBYB	AXBY
A**B	same	same
A*	A AX AXYZ	XAX
*A*B*	any name with a B the right hand side of an A	
*	all names	none
**	all names	none

If a library is specified before the pattern, the matching process applies to all members of the named library. If no library is specified, LIB is assumed.

The "not" convention can also be used in combination with the star convention. It indicates that the command is to process all members that are not matched by the pattern.

Example :

LIB :¬LIB3*

matches all members in LIB whose names do not begin with "LIB3".

The user can also specify the first and/or last name to be matched by the star convention based on alphabetical order. The first and last names need not be the names of members, these are only reference values for comparison.

IMPORTANT NOTE : THE FOLLOWING DOES NOT APPLY TO LIBMAINT SM

Examples :

- LM*, FROM = LMAB, TO = LMXY

means that all names beginning with LM, falling alphabetically between LMAB and LMXY (both inclusive) are to be processed.

- *XYZ, FROM = HHHXYZ
- ¬H*, FROM = G, TO = L
- *, FROM = A, TO = P
- *, TO = D;

Limited Star Convention

This limited form of the star convention has to be used :

- when the command has two positional parameters representing member names (Ex : COMPARE)
- when parameter NEW is used to create or update members under a new name

Limited star convention restricts the number of asterisks in the pattern to one occurrence. FROM and TO parameters may also be used in this limited form.

Examples :

- COMPARE INLIB1 : *, INLIB2 : *OLD

  denote that all members of INLIB1 are to be compared with the members of INLIB2 with the same name followed by OLD.

- MOVE INLIB1 : H*ERR, NEW = Z*WARN

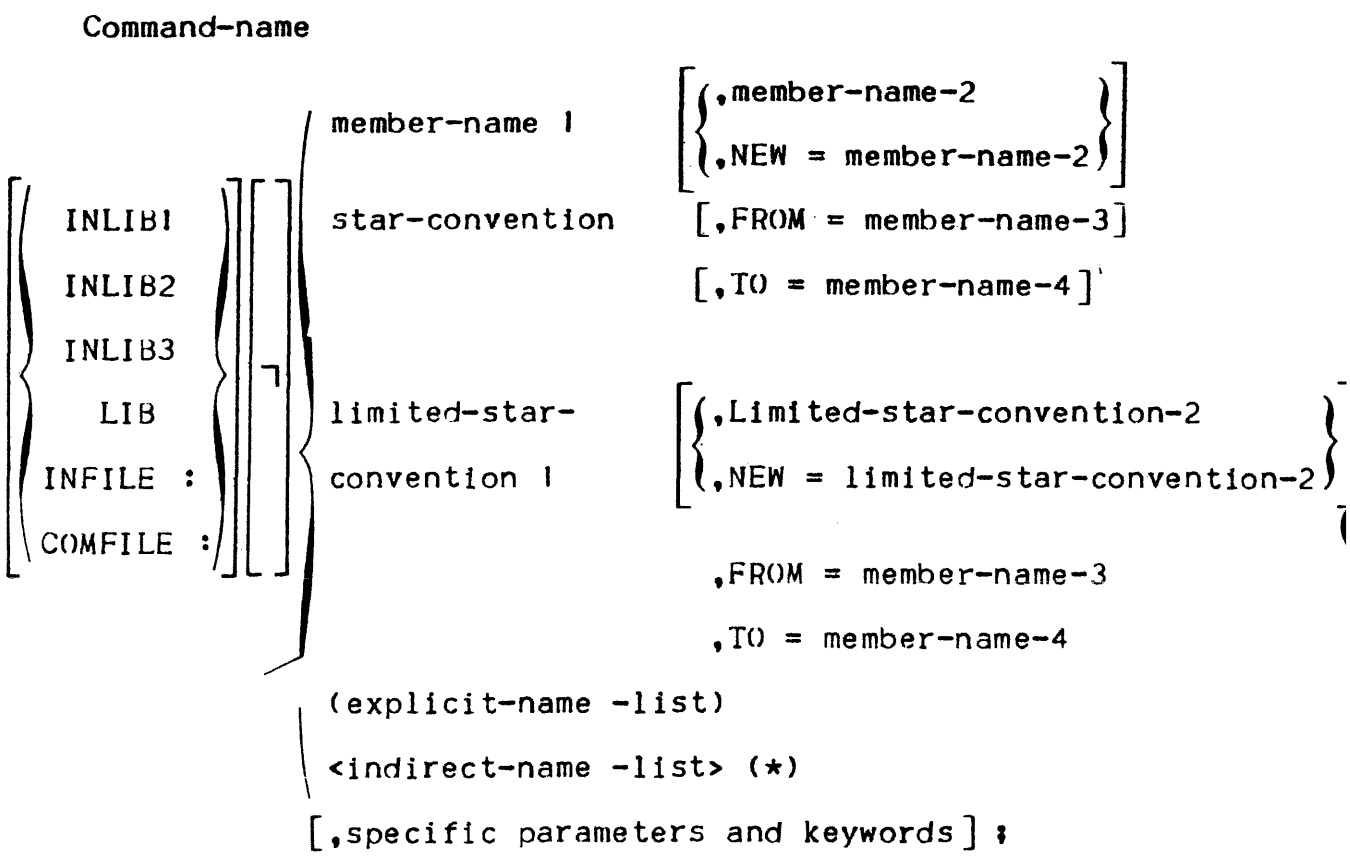
  applied to a library containing members :

  HMZIMZH
  HLIBERR
  HDBERR

will result in moving members HLIBERR and HDBERR into LIB and renaming them ZLIBWARN and ZDBWARN respectively.

GENERAL FORMAT OF COMMANDS

DOES NOT APPLY TO LIBMAINT SM



(*) APPLIES TO LIBMAINT SL ONLY

To improve the legibility of subsequent chapters, the following conventions will be used:

- "star-convention" will stand for "star-convention... FROM...TO..."

- "limited-star-convention" will stand for "limited-star-convention...FROM ....TO"

- commas will appear as separators between parameters and positional keywords. It is understood that these may be replaced by one or more spaces.

SECTION VII

COMMANDS APPLICABLE TO ALL LIBRARIES (Except SM)

Following is a description of LIBMAINT commands which are applicable to all types of libraries : SL, CU, LM (not applicable to SM).

Commands are sorted in alphabetical order and each description starts at the top of a page.

COMM

Function:

To introduce comment lines in the LIBMAINT execution report.

Format:

COMM Text of the comment;

Rules:

1. The text of the comment may span several input cards or records.
2. The text of the comment must be protected (i.e. enclosed between single quotation marks) if it contains a semi-colon.
3. The text of the comment may be parameterized (see EXEC).

Example:

COMM 'SAVE OF LIBRARIES ; LIST OF SAVED UNITS' ;

Function:

To skip to the top of a new page in the LIBMAINT report.

Format: EJECT ;

Rules:

This command is not printed out in the execution report.

ESCAPE

Function:

To submit an OCL statement for execution in interactive mode.

Format:

```
ESCAPE OCL-statement     ;
```

Examples:

```
ESCAPE DS X25 ;
ESCAPE SI TEST4:TESTLIB:C100:MS/M400:
```

Function:

To execute a sequence of LIBMAINT commands stored in a source library SSF unit. The executed commands may be parameterized.

Format:

```
EXEC  [ ( INLIB1 : )          ]  unit-name  [, VALUES = (p1,....., p99)]  ;
      [ ( INLIB2 : )          ]
      [ ( INLIB3 : )          ]
      [ ( LIB :    )          ]
```

Parameters:

- VALUES introduces a list of parameters which are to replace the corresponding &n in the executed unit ($1 \leq n \leq 99$). &1 will be replaced by p1, $2 by p2, ... &99 by p99.
  Acceptable forms for p1 are:

  - an unprotected character string which does not contain the characters space, ";" , "," , "'". Spaces preceding the first character or following the last significant character are ignored.

    Example: VALUES = (ABCD, EFGH, I)

  - a protected character string : i.e enclosed between single quotation marks "'". This string may contain any character but "'".

    Example: VALUES = ('STATUS ONLY ; ')

  - an empty protected string : i.e. two consecutive single quotation mark characters. The corresponding &n combination will be merely eliminated from the executed text.

    Example: VALUES = (A, ' ' , B)

  - an empty string : i.e. two consecutive commas, possibly separated by spaces. The corresponding &n combination will be left as it is.

    Example: VALUES = (A , , , D)

Rules:

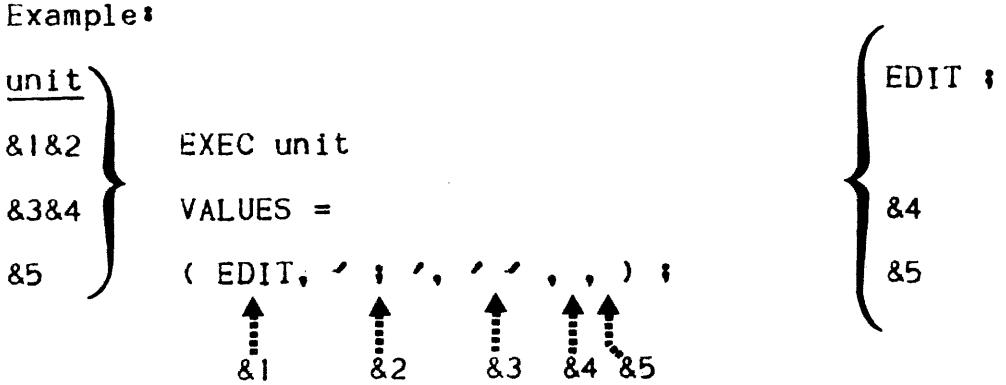
1 - EXEC may not appear inside an EXECuted sequence of LIBMAINT commands.

2 - Replacement of &n by the corresponding pn is effective in :

- LIBMAINT commands
- EDIT command's requests

it is ineffective in data introduced by a LIBMAINT command (MOVE COMFILE: , UPDATE) or by an input mode request of the text editor (EDIT's A, C, I).

Example:



Function:

To terminate a LIBMAINT Session. This command is optional, an end of file mark on COMFILE is treated as a QUIT command.

Format: QUIT ;

STATUS (also SM)

Function:

To continue or suspend the processing of LIBMAINT commands when an error has previously occurred.

Format:

```
STATUS  { EVEN  }
        { ONLY  }   ;
        { RESET }
```

Rules:

1. STATUS ONLY proceeds with the execution of the following command only if no error has previously occurred.
2. STATUS EVEN proceeds with the execution of the following command (and that command only) even if an error has occurred.
3. STATUS RESET resets the error count to zero.

Examples:

1.
```
DELETE A ;
STATUS EVEN ;
PRINT C ;
PRINT B
```
The PRINT Command will be executed even if the DELETE failed. But PRINT B will not be executed if either DELETE or PRINT C failed

2.
```
DELETE A ;
STATUS RESET ;
PRINT B ;
MOVE C ;
```
If the DELETE command fails, the following commands are executed.

Function:

To introduce a private title line after the standard LIBMAINT title lines in the execution report.

Format:

```
TITLE  [Text of the title];
```

Rules :

1. The text of the title can span several input cards or records. The maximum length is 240 characters.
2. The text must be protected between single quotation marks if a semi-colon appears in its body.
3. The text may be parameterized.
4. The TITLE command is not printed out on the execution report.
5. The new title text is taken into account from the first skip to top of page following the TITLE command.
6. If the title is to be printed at the first page of the LIBMAINT report, the first two commands must be: TITLE x; and EJECT;
7. An empty text (i.e. : TITLE ;) destroys the effect of a previous TITLE command.

Examples:

```
TITLE UPDATE OF SOURCE ;
TITLE 'TODAY ; 18 FEB 1978' ;
TITLE ;
```

SECTION VIII

COMMANDS APPLICABLE TO SL LIBRARIES

The following is a description of the LIBMAINT commands that are applicable to SL libraries.

A source language library contains a number of source units that may be:

- in System Standard Format (SSF)
- in Standard Access Record Format (SARF).

A source unit will typically contain a series of EBCDIC source text lines such as programs, documents, confidential information or any other kind of legible text. In normal mode, only SSF units are treated. However, some LIBMAINT commands apply to both (refer to the tables of section V) and conversion from one format to the other is possible (see MOVE command in this section) .

SSF units are prefixed with a special control record which is not directly user visible and which holds the relevant identification information for the unit, namely:

- Date and time of creation
- Date and time of last update
- Version number
- Language type
- Size in lines

This information is vital for the efficient handling of source language libraries and is automatically updated by LIBMAINT. Most commands display the identification of the units which they process. SSF source lines contain internal numbering which is left unchanged by all but the RENUMBER command (unless explicitly requested to do otherwise).

SARF source units have no line numbering and no information on their status or identification. SARF is a "raw" format.

Commands are presented in alphabetical order and each description starts at the top of a page.

CODE

Function:

To provide maximum security for data stored in SSF source unit(s), this command permits the coding of the units contents according to a key which need not be stored in the system.

Format:

```
                 [(INLIB1 :)]   [ ]   (member-name-1,  NEW = member-name-2)
                 [(INLIB2 :)]   [¬]   (star-convention                      )
CODE             [(INLIB3 :)]         (limited-star-convention-1           )
                 [        :)]         [, NEW = limited-star-convention-2]
                 [(LIB    :)]   [ ]   (explicit-name-list)
                                      <indirect-name-list>

        , KEY = 'keyvalue'   [, REPLACE ] ;
```

Parameters:

- KEY is a protected string of no more than 16 non-blank characters which is used for the coding. This parameter is mandatory.

- NEW specifies the name(s) of the resulting member(s). This may be used with member-name or limited-star-convention.

- FROM, TO may be used with star-convention and limited-star-convention.

Keywords:

- REPLACE allows the member-name being written in LIB to replace (over-write) a member with the same name previously present in library LIB.

  - if the specified or selected library is LIB and if NEW is not specified, REPLACE need not be specified.

  - REPLACE may be used even if the LIB library contains no unit with the resulting member-name.

Rules :

1. When member-name is specified, search rules apply unless a library keyword is used.
2. When star convention, explicit or indirect name list is used, default library is LIB, unless a library keyword is specified.
3. The result of the coding is in library LIB.
4. See DECODE command for the reverse process.
5. This command may be parameterized.

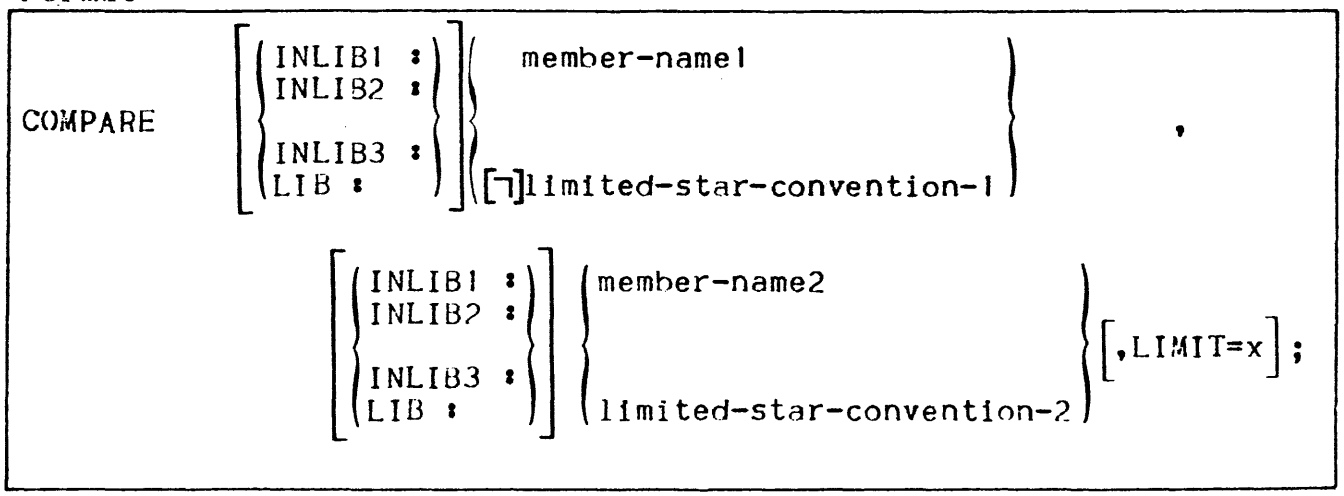
Examples:

```
CODE INLIB1 : SOURCE_PROC, NEW = CSOURCE_PROC, REPLACE,
KEY = '+*AW';

CODE * , KEY = 'MYKEY' ;
```

Function:

Compare two SSF source units and print out the changes made to the first argument unit to yield the second argument unit.

Format:



Parameters:

- LIMIT specifies the maximum number of discrepancies to be taken into account before processing stops. Default value is 50; maximum value is 32767.

- FROM, TO may be used with limited-star-convention.

Rules:

1. When member-name-1 or member-name-2 is specified, search rules apply unless a library keyword is used.

2. When limited-star-convention is used, default library is LIB unless a library keyword is used.

3. The output is organized with the assumption that the first unit was edited, resulting in the second.

4. Both compared members must be in SSF.

5. Internal line numbering is not taken into account in the comparison process. Thus, two members which differ only by their numbering will be considered as identical.

6. The command may be parameterized.

Examples:

- COMPARE H_LIB_E1, H_LIB_E1BIS, LIMIT = 10 ;

  H_LIB_E1 and H_LIB_E1BIS are compared until 10 discrepancies are found. Search rules apply for both arguments.

- COMPARE INLIB1 : H_LIB_ELIST, INLIB2 : H-LIB_ELIST ;

  H_LIB_ELIST in libraries INLIB1 and INLIB2 are compared. No search rule applies. Comparison will terminate if more than 50 discrepancies are found.

- COMPARE INLIB1 : H_LIB_*, INLIB2 : H_LIB_*_OLD, LIMIT = 1 ; will compare

  INLIB1 : H_LIB_ERR with INLIB2 : H_LIB_ERR_OLD

  INLIB1 : H_LIB_GO with INLIB2 : H_LIB_GO_OLD etc...

Function:

Create a SSF unit containing a complete or selective list of the member names of a library or a list of user specified names.

Format:

```
         [ ( INLIB1 : ) ] [   ] ( member-name-1          )
         [ ( INLIB2 : ) ] [   ] ( star-convention        )
CRLIST   [ (          ) ] [ ¬ ] (                        )
         [ ( INLIB3 : ) ] [   ] ( (explicit-name-list)   )
         [ ( LIB :    ) ] [   ] ( <indirect-name-list>   )
  , member-name-2  [, NUMBER  [= (n1  [, n2 ])]][, REPLACE ]:
```

Parameters:

- FROM, TO may be used with star-convention

Keywords:

- NUMBER specifies that the lines of the created (member-name-2) SSF unit are to be numbered, beginning with n1, and incrementing by n2 for each new line. Default values are n1 = 10 and n2 = 10.

- REPLACE allows the created member name to replace (overwrite) an existing member with the same name in LIB. This may be used even if no such member exists in LIB.

Rules:

1. One record (line) is created in member-name-2 (in LIB) for each member-name specified in the command or found in the input library. The record contains the member name followed by a single space character.

2. Member-name-2 may be later used as an indirect name list by another LIBMAINT command

3. When star-convention is specified the directory of the specified or implicit (LIB) library is read and matching names stored in member-name-2 of LIB.

4. When member-name-1, explicit or indirect name list, not preceded by "¬" are used, library keyword is immaterial.

5. The command may be parameterized.

Examples:

- This command will be best appreciated when one has to handle a set of members, the list of names of which cannot be easily obtained by star convention or explicit name list. Select a superset of the required names in a CRLIST command, then eliminate in the resulting unit all undesirable names. The unit may then be used as an indirect name list.

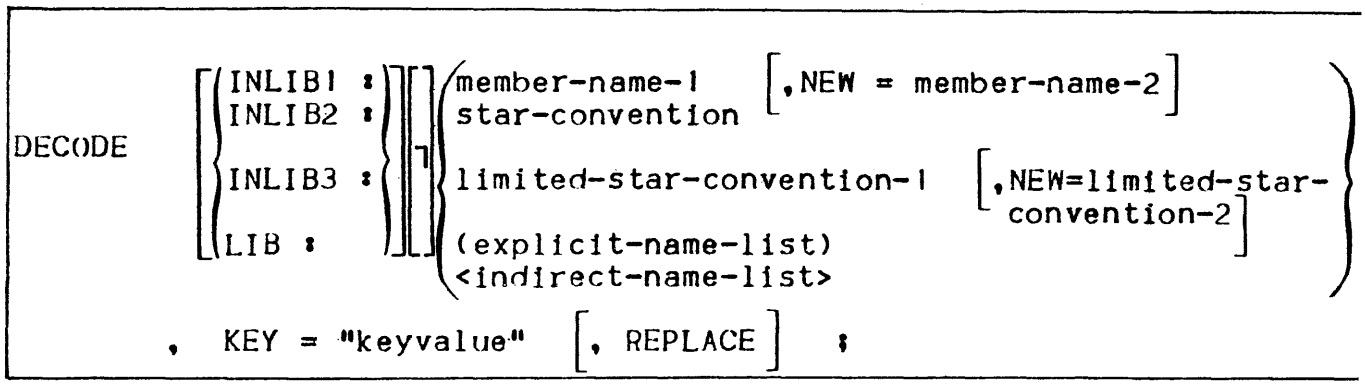
- CRLIST is also useful when a series of commands is to be applied to the same set of members. One can then proceed as follows:

```
CRLIST (list-of-the members) , MYLIST ;

command 1 ... < MYLIST > ..... ;

.
.
.
command n ... <MYLIST> ....;

DELETE MYLIST ;
```

Function:

Reconstruct the original unit, given a coded unit and its coding key.

Format:



Parameters:

- KEY is a protected string denoting the key which was used to code the unit. It is a string of no more than 16 non-blank characters. This parameter is mandatory.

- NEW Specifies the name to be given to the reconstructed member. NEW may be used with member-name-1 or limited-star-convention.

- FROM, TO may be used with star-convention and limited-star-convention.

Keywords:

- REPLACE allows the reconstructed member to replace (overwrite) an existing unit in LIB.

  - if the specified or selected library is LIB and NEW is not specified, REPLACE need not be present.

  - REPLACE may be used even if no member with the same name exists in LIB.

Rules:

1. When member-name is specified, search rules apply unless a library keyword is used.

2. When star-convention, explicit or indirect name list is used, default library is LIB unless a library keyword is specified.

3. The reconstructed member is written in LIB.

4. See CODE command for the reverse operation.

5. The command may be parameterized

Examples:

```
DECODE LIB : CSOURCE_PROC, NEW = SOURCE_PROC,
REPLACE, KEY = ' + SAW ' ;
DECODE LIB : * , NEW = K_* , KEY = ' MYKEY ' ;
DECODE * , KEY = ' @ XZK ' ;
```

DELETE

Function:

Delete one or more units in a source unit library LIB.

Format :

```
DELETE [LIB :] [¬]  ⎧member-name          ⎫
                    ⎪star-convention      ⎪   ;
                    ⎪                     ⎪
                    ⎪(explicit-name-list) ⎪
                    ⎩<indirect-name-list> ⎭
```

Parameters :

- FROM, TO may be used with star-convention

Rules:

1. No search rule applies : member(s) must exist in library LIB.
2. The only allowed library keyword is LIB.
3. Deleted members may be in SSF or SARF format.
4. Identification of deleted members is displayed for SSF members.
5. The command may be parameterized.

Examples:

```
DELETE LIB: *A , FROM = ALPHA, TO = OMEGA ;

DELETE SOURCE ;

DELETE H*J ;
```

Function: invoke the text editor using the requests following in the command input stream.

Format: EDIT ;

Rules: See section XIII.

GLOBAL EDIT

Function:

Repeatedly invoke the TEXT EDITOR for a series of members defined in the EDIT command.

Format:

```
                  [ (INLIB1 ;) ]   [ ¬ ]   ( member-name-1 [, NEW = member-name-2] )
                  [ (INLIB2 ;) ]           ( star-convention                       )
EDIT              [ (INLIB3  ) ]           ( limited-star-convention-1             )
                  [            ]           ( [, NEW=limited-star-convention-2]     ) ;
                  [ (LIB    ;) ]           ( (explicit-name-list)                  )
                                           ( <indirect-name-list>                  )
```

Parameters:

- NEW Specifies the name(s) of the unit(s) resulting from the W or Z requests (see section XIII). This may be used with member-name or limited-star-convention.

- FROM, TO may be used with star-convention or limited-star-convention.

Rules:

1. Requests for GLOBAL EDIT are identical to those for EDIT (see section XIII) except for extensions of R, W and Z. Such requests that refer to units defined in the command must take one of the following forms :

   R [INLIB1 : / INLIB2 : / INLIB3 : / LIB :] &0

   { W / Z } [(type)] &0

   When a library keyword is used with the R request it must be the same as the one in the EDIT command.

2. If NEW is used and member-name-2 exists in the LIB library, the corresponding W &0 will be rejected, Z &0 will be accepted to overwrite the member.

3. &0 is reserved for R, W and Z requests. It is replaced by the corresponding member-name in the command.

Example :

```
EDIT INLIB1 : H_*_ERR, NEW = I_*_WARN :

R INLIB1 : &0
.
.
.
requests
.
.
.
.
Z &0

Q
```

If INLIB1 contains members : H_MZ_IMZH
H_LIB_ERR
H_LK_EDIT
H_DB_ERR

requests will apply to

H_LIB_ERR to create an edited unit named I_LIB_WARN and
H_DB_ERR to create an edited unit named I_DB_WARN.

Function:

Improve the readability of a GPL source unit by indenting it according to a set of rules described below.

Format:

```
INDENT  [ {INLIB1 :} ] [ [ ┐ ] ] { member-name-1    , [NEW = member-name-2]           }
        [ {INLIB2 :} ]           { star-convention                                    }
        [ {INLIB3 :} ]           { limited-star-convention-1 [, NEW=limited-          }
        [ {        } ]           {                             star-convention-2]     }
        [ {LIB :   } ]           { (explicit-name-list)                               }
                                 { <indirect-name-list>                               }

        [,CM = xxx] [, IN = yy] [, LM = zz] [, REPLACE] ;
```

Parameters:

- NEW Specifies the name(s) of the resulting member(s); this may be used with member-name and limited star-convention.
- FROM, TO may be used with star-convention or limited star-convention.
- CM Sets the comment column to position xxx. Comments are lined up in this column unless they begin a line and are preceded by a blank line (or are at the beginning of the program or are a comment beginning in column 1). If this parameter is omitted, default for xxx is 61.
- IN Sets the value of indentation for each level to yy. Each DO, BEGIN, etc... statement will cause additional yy spaces of indentation until the matching statement is encountered. If this parameter is omitted, default for yy is 5.
- LM Sets the left margin (indentation for normal program statement) to zz. If this argument is omitted, default value for zz is 11.

Keywords:

- REPLACE Allows the member(s) being written to the output library LIB to replace (overwrite) existing member(s) with the same name.
  - If the specified or selected library is LIB and NEW is not used, REPLACE need not be stated.
  - REPLACE may be used even if the resulting member(s) do not exist in LIB.

Rules :

1. When member-name is specified, search rules apply unless a library keyword is stated.

2. When star-convention, explicit or indirect name list is used, default library is LIB unless a library keyword is specified.

3. The resulting member(s) are obtained in LIB or OUTFILE depending on LIBMAINT assignments.

4. Declaration statements are indented five spaces. Structure declarations are indented according to structure level number. After level 2, additional levels are indented two more spaces each.

5. Multiple spaces are replaced by a single space, except within strings or non-leading spaces in comments. Spaces are inserted before a left parenthesis, after commas, and around the constructs = , -> , < = , > = , and ¬=. Spaces are deleted when found after a left parenthesis or before a right parenthesis.

6. Parentheses are counted and are expected to balance at every semi-colon. If parentheses do not balance at a semi-colon, or if the unit terminates in a string or in a comment, a diagnostic is produced.

7. The command may be parameterized.

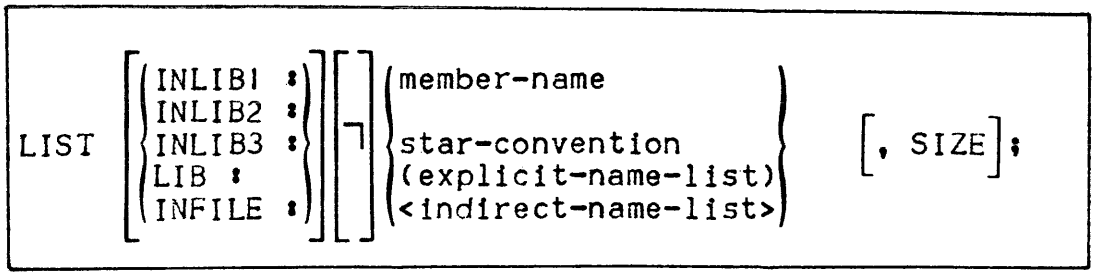
Examples:

```
IDENT INLIB3 : PROG, NEW =IPROG, CM = 81, LM = 7, IN = 3 ;

INDENT LIB : * , FROM = ALPHA, TO = OMEGA,

NEW = *NEW REPLACE, CM = 71 ;
```

Function:

Produce a complete or selective table of contents of a source language library or SSF sequential file.

Format:



Parameters:

- FROM, TO may be used with star-convention

- SIZE When specified, allocated space is displayed for each member (unit = 1 block)

Rules:

1. When member-name is specified, search rules apply unless a library keyword or INFILE is used.

2. When star-convention, explicit or indirect name list is specified, default library is LIB unless a library keyword or INFILE is used.

3. For a library, members are listed in alphabetical order.

4. For a sequential file, members are listed in storage order.

5. For a library, occupancy information is displayed.

6. Identification of SSF units is displayed

7. The command may be parameterized

Examples:

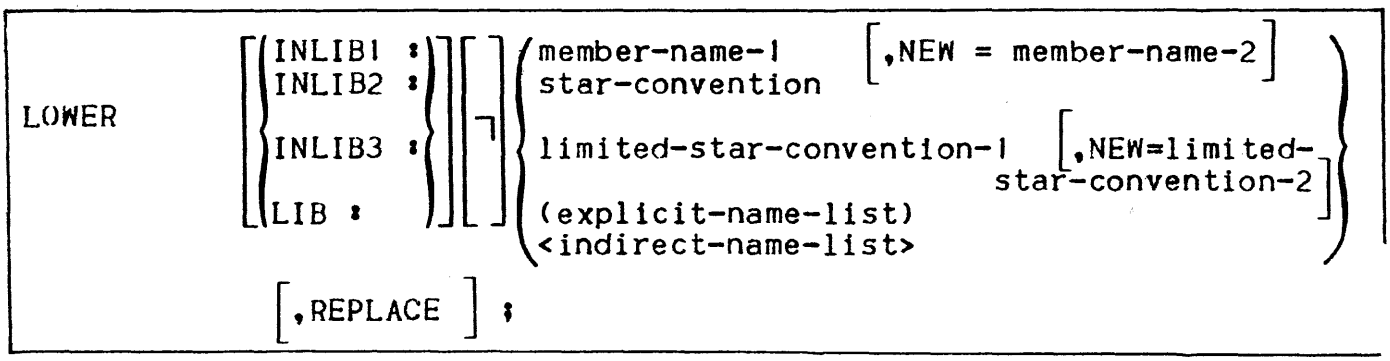
```
LIST INFILE : MEMBER_*_NEW ;

LIST * , FROM = ALPHA, TO = OMEGA ;
```

LOWER

Function:

Convert source language unit(s) into lower case letters.

Format:



Parameters:

- NEW Specifies the new name(s) to be given to the converted unit(s).

- FROM , TO May be used with star-convention or limited-star-convention.

Keywords:

- REPLACE Allow the unit(s) being written in output library LIB to replace (overwrite) an existing unit with the same name.

  - if the specified or selected library is LIB, and if NEW is not specified, REPLACE need not be stated.

  - REPLACE may be used even if no member with the same name exists in LIB.

Rules:

1. When member-name is used, search rules apply unless a library keyword is specified.

2. When star-convention, limited-star-convention, explicit or indirect name list is used, default library is LIB unless a library keyword is specified.

3. The command may be parameterized.

Examples: LOWER LIB : MYMEMBER ;

```
LOWER INLIB1 : <INLIB2 : MYLIST> ;
```

Function:

- Move one member from COMFILE to a library.

- Move one or more members from a library to the LIB library or to a sequential file.

- Move one or or more members from a sequential file to a library or another sequential file.

- Move a SSF member from a library to the LIB library or to a sequential file with SARF format (SSF to SARF);

- Move a SARF member form a library or sequential file to the LIB library with SSF format (SARF to SSF).

Format:

```
          [ { INLIB1 :   } ] [ ] { member-name-1  [, NEW = member-name-2] }
          [ { INLIB2 :   } ] [ ] { star-convention                        }
          [ { INLIB3 :   } ] [ ] {                                        }
MOVE      [ { LIB :      } ] [¬] { limited-star-convention-1  [,NEW =     }
          [ {            } ] [ ] {       limited-star-convention-2]       }
          [ { COMFILE :  } ] [ ] { (explicit-name-list)                   }
          [ { INFILE :   } ] [ ] { <indirect-name-list>                   }

[, TYPE      = language-type] [, NUMBER [= (n1  [, n2] )]]
               [, END = "string"]

[, INFORM =  {SSF }]   [, OUTFORM =  {SSF }] [,REPLACE] [,CHECK]
             {SARF}                  {SARF}

[, ENDCHAR= "char1"] [,CONTCHAR = "char1"] [FORMAT=(n1,n2,n3,n4 [,n5] )] ;
```

Parameters:

- NEW Specifies the name(s) of the resulting member(s). This may be used with member-name or limited-star-convention. NEW cannot be used with COMFILE : or INFILE : with SARF.

- FROM, TO may be used with star-convention or limited-star-convention.

- TYPE determines the language type of the output unit. This type will be permanently associated with the unit and be part of

its identification. The language type determines the default FORMAT (see below) of input records when the user transfers units from COMFILE, input libraries or INFILE in SARF format. In these cases TYPE is mandatory. It is possible to use parameter FORMAT to override the default values provided. The following table gives these default values (refer to FORMAT, below for explanations).

TABLE 8-1. DEFAULT FORMATS

Language-type	Means	Default FORMAT				
		n1	n2	n3	n4	n5
FOR or FORTRAN	FORTRAN	0	0	1	72	0
CBX or COBOLX	Extended COBOL	1	6	7	72	0
COB or COBOL	COBOL	1	6	7	80	0
RPG	Unified Report Program	0	0	1	80	0
JCL	Job Control Language	0	0	1	80	0
DAT or DATASSF	Data in SSF format	0	0	1	80	0
DEL or HUDEL	Honeywell User Document Entry Language	0	0	1	80	0
GPL	GCOS Programming Language	1	6	7	80	0

- NUMBER Specifies that the line numbers (if any) in the input records are to be discarded and that the lines be numbered beginning at n1 and incrementing by n2 at each new input record. Default values are n1 = 10 and n2 = 10.

- END Denotes the marker to be used to specify the end of a sequence of input cards in COMFILE.

  - maximum length of string is 8 characters

  - all characters except single quote mark are allowed

  - if END is omitted, implicit value for string is //EOD

  - string or //EOD is mandatory when COMFILE is used; it must begin in column 1 of the input record.

- FORMAT This parameter is used to override default values derived from the language type or the TYPE parameter. It defines the position of the line number, of the text and vertical form control fields in the record. This is defined by means

of 5 values denoted as n1, n2, n3, n4 and n5.

- Line number field expands from columns n1 to n2. Zero values denote that no line number is present. The line number must be right justified in the field. Blanks are treated as zeroes.

- Text field expands from columns n3 to n4. It may overlap the line number field.

- n5 is used to specify the column in which a particular form control option is coded. Zero value denotes no form control option. Applicable codes are:

Space	vertical space 1 line before this line
0	double space before this line
-	triple space before this line
+	do not vertically space before this line
1	Skip to top of page with this line.

- Character * may be used to denote variable length fields. If n2 = *, line number terminates at first non-digit character. If n3 = *, text field begins with the first non-digit character on input record except when this first non-digit character is a space. If n4 = *, text continues until end of record. Neither n1 nor n5 may be *.

- CONTCHAR The character defined by parameter CONTCHAR is considered as a continuation mark if it is the last significant character of an input card or of a SARF record. This option may be used with MOVE with INFORM = SARF ; it may be used simultaneously with ENDCHAR (see below).

- ENDCHAR The character defined by parameter ENDCHAR is considered as an end of record delimiter if it is the last significant character of an input card or of a SARF record. This option may only be used with MOVE with INFORM = SARF ; it may be used simultaneously with CONTCHAR (see above). When ENDCHAR is specified, all input records must terminate with the specified character.

- INFORM Specifies that input member is in SARF or SSF.

  Default is SSF.

- OUTFORM Specifies that output member is to be SARF or SSF.

  Default is SSF. When OUTFORM = SARF, line numbers are inserted in the text in accordance with the standard format corresponding to the members type (if any). This format may be overridden by parameter FORMAT (see above).

Keywords:

- REPLACE Allow the member(s) being written in output library LIB to replace (overwrite) member(s) with the same name previously existing in LIB.

  - When assigned LIBMAINT output is OUTFILE, REPLACE is ignored and command will operate regardless of whether REPLACE permission is given or not.

  - REPLACE may be used even if no member with the resulting name is present in LIB.

-CHECK Specifies that it is to be checked that line numbers (for type = GPL, COB, CBX) are in non-descending order. This option can be used with MOVE COMFILE and MOVE INFILE SARF.

Rules:

1. When member-name is specified, search rules apply unless a library keyword, COMFILE or INFILE is used.

2. When star-convention, explicit or indirect name list is used, default library is LIB unless a library keyword or INFILE is specified.

3. Library keyword LIB cannot be specified when LIBMAINT assigned output is OUTFILE.

4. The command may be parameterized.

Examples:

1. <u>To store a whole unit read from cards into a library or sequential file</u>

```
MOVE COMFILE : member-name [, FORMAT ...] [, CHECK]
[, OUTFORM = ...] [, NUMBER = ...] , TYPE = ....
[, END = 'string'] [, REPLACE] [, CONTCHAR = ...]
[, ENDCHAR = ... ] ;
```

- Output may be SARF or SSF

- Records of more than 80 characters can be created by means of CONTINUE or RECEND. The maximum length of such records is however limited to 256.

- MOVE COMFILE : S1, TYPE = DATASSF, END = 'FIN' , REPLACE ;

```
.
.
.
deck of cards
.
.
.
FIN
```

2. <u>To transfer SSF units from a library or file into a library or file.</u>

```
         [ {INLIB1 : } ] [   ] {member-name1  [, NEW = .....]                  }
         [ {INLIB2 : } ] [   ] {star-convention                                }
MOVE     [ {INLIB3 : } ] [ ┐ ] {limited-star-convention [, NEW = ....]         }
         [ {INFILE : } ] [   ] {(explicit-name-list)                           }
         [ {LIB :    } ] [   ] {<indirect-name-List>                           }

           [, REPLACE]  [, NUMBER] ;
```

Output is LIB or OUTFILE depending on the LIBMAINT assignment.

```
MOVE INLIB1 :┐* OLD

MOVE TEST4, NUMBER, REPLACE ;

MOVE INFILE : * ;

MOVE INFILE : UNIT1, NEW = UNIT2, NUMBER, TYPE = COBOL ;

MOVE INLIB1 : *, FROM = ALPHA, TO = OMEGA;

MOVE INLIB3 : UNIT (1, 2, 3, 4), NUMBER ;

MOVE <INLIB2 : MYLIST> , REPLACE ;
```

3. <u>To transfer SARF from INFILE into LIB or OUTFILE.</u>

```
MOVE INFILE : member-name, INFORM = SARF, TYPE = ... ,

[, OUTFORM = ...] [, CHECK] [, REPLACE] [, FORMAT]

[, CONTCHAR = ... ] [, ENDCHAR = ...] ;
```

- Output is LIB or OUTFILE depending on LIBMAINT assignment.

- Output may be in SSF or SARF (OUTFORM parameter)

- MOVE INFILE : UNIT1, TYPE = DATASSF, FORMAT = (0, 0, 1, *),

  INFORM = SARF ;

4. <u>To transfer SSF or SARF from a library into LIB or OUTFILE in SARF</u>

```
         [ {INLIB1 : } ]   {member-name          }
MOVE     [ {INLIB2 : } ]   {star-convention      } [, FORMAT = ...]
         [ {INLIB3 : } ]   {(explicit-name-list) } [, NUMBER = ...]
         [ {LIB :    } ]   {<indirect-name-list> }

            [, TYPE = .... ] , OUTFORM = SARF [, INFORM = ....] ;
```

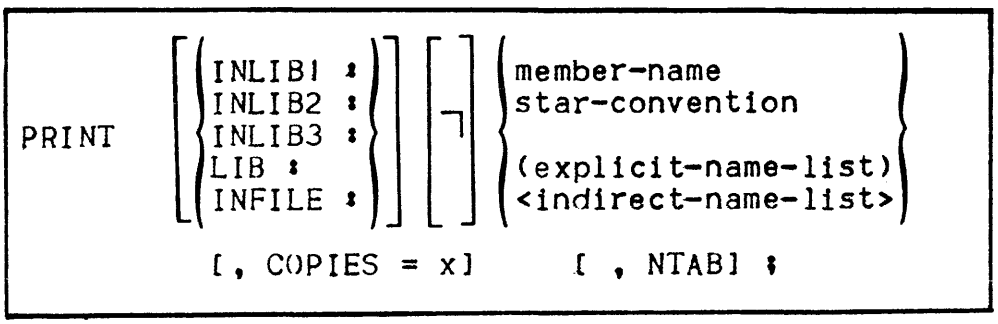
- FORMAT describes the output format.

```
- MOVE UNIT1, OUTFORM = SARF, TYPE = COBOL ;

- MOVE INLIB2 : UNIT1, OUTFORM = SARF, TYPE = CBX, FORMAT =
  (1,4,7,97) ;

- MOVE UNIT1, OUTFORM = SARF, FORMAT = (1, 6, 7, *) ;
```

Function:

Print the contents of one or more members of a source unit library or sequential file.

Format:



Parameters:

- FROM, TO may be used with star-convention

- COPIES Specify the number of copies to be produced ($1 \leq x \leq 9$)

Keywords:

- NTAB : Do not expand tabulation characters (hexadecimal 05)

Rules:

1. Search rules apply when no library keyword nor INFILE is specified.

2. When star-convention, explicit or indirect name list is specified, default library is LIB, unless a library keyword or INFILE is specified.

3. Selection of a member or set of members from INFILE is possible.

4. The command may be parameterized.

Examples:

```
PRINT INFILE : ¬<LIST> , COPIES = 3 ;
PRINT * ;
```

Function:

Punch one or more members of a source unit library or sequential file in SSF format.

Format:

```
PUNCH  [ { INLIB1 : }  ]  [ ¬ ]  { member-name                  }
       [ { INLIB2 : }  ]          { star-convention             }
       [ { INLIB3 : }  ]          {                             }
       [ { LIB :    }  ]          { (explicit-name-list)        }
       [ { INFILE : }  ]          { <indirect-name-list>        }

    [, TYPE = language-type]   [, ENDCHAR = 'char'] ;
```

Parameters:

- FROM, TO may be used with star-convention

- ENDCHAR the specified character is to be punched as the last character of each output record.

- TYPE Determine the format in which cards are to be punched. Basically, TYPE indicates the positions in which the line numbers are to be punched (if any) and the width of the punched text. PUNCH is such that the punched cards will be accepted as such by the MOVE COMFILE command with the same TYPE parameter. When no type is specified, the stored unit type is assumed. The following table summarizes the punching conventions according to the language type indicated.

TABLE 8-2. PUNCHING CONVENTIONS

Language type	Means	Line number	Text
COBOL or COB	Standard COBOL	1 to 6	7 to 80
COBOLX or CBX	Extended COBOL	1 to 6	7to 72
FORTRAN or FOR	FORTRAN	None	1 to 72
RPG	Unified Report Generator Language	None	1 to 80
JCL	Job Control Language	None	1 to 80

TABLE 8-2. (CONT). PUNCHING CONVENTIONS

DATASSF or DAT	System Standard Format Data	None	1 to 80
GPL	GCOS Programming Language	1 to 6	7 to 80

Rules :

1. If the specified member has no associated type and TYPE parameter is not supplied, an error occurs.
2. When member-name is specified, search rules apply unless a library keyword or INFILE is specified.
3. When star-convention, explicit or indirect name list is used, default library is LIB unless a library keyword or INFILE is specified.
4. The line number and text are punched according to the unit's language type or the supplied TYPE parameter.
5. Records longer than 80 characters are truncated, unless ENDCHAR is specified.
6. The command may be parameterized.

Examples:

```
PUNCH INLIB1 : HLIB* , TYPE = COBOL ;

PUNCH * ;
```

RENAME

Function:

Rename one or more members of a source unit library in SSF or SARF format.

Format:

```
RENAME [LIB :] { Member-name1, [NEW =] member-name-2
               { [-]limited-star-convention-1, [NEW =] limited-
                                               star-convention-2 } ;
```

Parameters:

- NEW Specify the new name(s) of the renamed member(s).

- FROM, TO may be used with star-convention or limited-star-convention.

Rules:

1. New member name(s) must not exist on LIB ; REPLACE parameter is not allowed.

2. Renamed member(s) must be present in LIB.

3. After the command, the old name(s) no longer exist.

4. The command may be parameterized.

Examples:

```
RENAME HLIBX3 , NEW = LIB_3 ;

RENAME LIB : H*ERR, I*WARN ;
```

Function:

Renumber one or more source members in SSF format into LIB.

Format:

```
RENUMBER [LIB :] [¬]    { member-name                  }
                        { star-convention             }
                        { (explicit-name-list)        }
                        { <indirect-name-list>        }
            [NUMBER  [= (n1 [, n2 ])]    ;
```

Parameters:

- FROM, TO may be used with star convention.

- NUMBER specify the initial value and the increment used for numbering. n1 is the initial value, n2 is the increment. Default value is 10 for both n1 and n2.

Rules:

1. Members must be present in library LIB.
2. The command may be parameterized.

Examples:

```
RENUMBER UNIT1, NUMBER = (1, 1) ;

RENUMBER UN* , NUMBER = (20, 20) ;

RENUMBER UNIT (1, 2, 3, 18, 27) ;

RENUMBER * FROM = H , TO = Y ;
```

Function:

Sort the lines of a SSF source unit or units in ascending or descending EBCDIC collating sequence.

Format:

```
                [ [ INLIB1 : ] ]   [ ]  { member-name1   [, NEW = member -name-2] }
                [ [ INLIB2 : ] ]        { star-convention                         }
SORT            [ [ INLIB3 : ] ]   ¬    { limited-star-convention1  [, NEW =       }
                [ [          ] ]        {                  limited-star-convention-2] }
                [ [ LIB    : ] ]        { (explicit-name-list)                    }
                                        { <indirect-name-list>                    }

                { [, ASC ]  }
                { [, DESC]  }   [,REPLACE][SORTKEY = (n1 [,n2])]
                                [, NUMBER [ = (n1 [,n2] ) ] ] ;
```

Parameters:

- NEW Specify the name(s) to be given to the resulting unit(s). It may be used with member-name or limited-star-convention.

- FROM, TO May be used with star convention or limited star convention

- SORTKEY - n1 gives the position of the first character of the sort key in the record (line).

  - n2 maximum length of the sort key. If omitted,

    n2 = 256 - n1 is assumed.

  - if SORTKEY is omitted, default values taken are

    n1 = 1 and n2 = 255 (i.e. the key is the whole record).

- NUMBER specifies that the lines of the created (member-name-2) SSF unit are to be numbered, beginning with n1, and incrementing by n2 for each new line. Default values are n1 = 10 and n2= 10.

Keywords:

- ASC Sort in ascending order.

- DESC Sort in descending order.

- REPLACE Allow the member-name(s) being written to output library LIB to replace (overwrite) an existing unit with the same name in LIB.

  - if the selected or specified library is LIB and if NEW is not specified, REPLACE need not be specified.

  - REPLACE may be used even if the resulting member(s) do not exist in LIB.

Rules:

1. When member-name is used, search rules apply unless a library keyword is specified.

2. When star-convention, explicit or indirect name list is specified default library is LIB unless a library keyword is given.

3. The resulting member(s) are output on LIB.

4. The command may be parameterized.

5. SORT is restricted to sorting reasonably small units (a few thousand lines is a maximum). For other applications the GCOS SORT utility should be used.

Examples:

```
SORT INLIB2 : MYS1, DESC, SORTKEY = (1, 109) ;

SORT INLIB1 : HLIB* , NEW = ZLIB* , REPLACE ;
```

Function:

Modify the contents of a source unit according to the requests following the commands in the command input stream. UPDATE provides limited editing facilities.

Format:

```
                [(INLIB1 :)]   [ ]   {member-name-1   [,NEW = member-name2]  }
                [(INLIB2 :)]   [¬]   {star-convention                         }
UPDATE          [(INLIB3 :)]   [ ]   {limited-convention-1 [,NEW = limited-   }
                [(       )]   [ ]   {                 star-convention-2]       }
                [(LIB :  )]   [ ]   {(explicit-name-list)                     }
                                    {<indirect-name-list>                     }

[, NUMBER [= (n1 [, n2 )]] [, TYPE = language-type]
[, END = 'string'] [, FORMAT = (n1, n2, n3, n4 [, n5] )]
[, REPLACE][, CONTCHAR = 'char1'][, ENDCHAR = 'char1'] ;
```

-NEW Specifies the name(s) of the resulting member(s). This may be used with member-name or limited star-convention.

-FROM, TO May be used with star-convention or limited star-convention.

- TYPE Determines the type of the output unit. This type is permanently associated with the unit and is part of its identification. The language type determines the default FORMAT (see below) of the request following the command. When TYPE is not specified, the type of the updated unit is taken. It is possible to use parameter FORMAT to override the default values provided. The following table gives these default values (refer to FORMAT below for explanations).

TABLE 8-3. DEFAULT VALUES

Language-type	Means	Default FORMAT				
		n1	n2	n3	n4	n5
FOR or FORTRAN	FORTRAN	0	0	1	72	0
CBX or COBOLX	Extended COBOL	1	6	7	72	0
COB or COBOL	COBOL	1	6	7	80	0
RPG	Unified Report Program Generator	0	0	1	80	0

TABLE 8-3 (CONT). DEFAULT VALUES

JCL	Job Control Language	0	0	1	80	0
DAT or DATASSF	Data in SSF format	0	0	1	80	0
GPL	GCOS Programming Language	1	6	7	80	0

-NUMBER Specifies that the resulting unit(s) are to be renumbered after update. The initial value of the number is n1, the increment at each new line is n2. Default values are n1 = 10, n2 = 10.

-END Denotes the marker to be used to specify the end of the requests.

- maximum length of string is 8 characters.
- all characters except single quote mark are allowed
- if END is omitted, implicit value for string is //EOD
- string or //EOD is mandatory. It must begin in column 1 of the record following the last update request.

- FORMAT This parameter is used to override default derived from the language type or the TYPE parameter. It defines the position of the line number of the text and vertical form control fields in the record. This is defined by means of 5 values denoted as n1, n2, n3, n4 and n5.

- Line number field expands from columns n1 to n2. Zero value denotes that no line number is present. The line number must be right justified in the field. Blanks are treated as zeroes.
- Text field expands from columns n3 to n4. It may overlap the line number field.
- n5 is used to specify the column in which a particular form control option is coded. Zero value denotes no form control option. Applicable codes are:

  Space: vertically space 1 line before this line.

  0 : double space before this line

  - : triple space before this line

  + : do not vertically space before this line

  1 : skip to top of page with this line.

- Character * may be used to denote variable length fields.

  If n2 = *, line number terminates at first non-digit character.
  If n3 = *, text field begins with the first non-digit character on input record except when this first non-digit character is a space.

  If n4 = *, text continues until end of record. Neither n1 nor n5 may be *.

- CONTCHAR The character defined by parameter CONTCHAR is considered as a continuation mark if it is the last significant character of an update card. This option may be used simultaneously with ENDCHAR (see below).

- ENDCHAR The character defined by parameter ENDCHAR is considered as an end of record delimiter if it is the last significant character of an update card. This option may be used simultaneously with CONTCHAR (see above). When ENDCHAR is specified, all update cards must terminate with the specified character.

Keywords:

-REPLACE Allow the updated member(s) to replace (overwrite) member(s) with the same name.

- if the specified or selected library is LIB and NEW is not stated, REPLACE need not be specfied.

- REPLACE may be used even if no previous member with the resulting name exists in LIB.

Rules:

1. Member-name-1 is the name of the member to which the modification requests are to be applied. This member is found by means of the search-rules, unless a library keyword is specified.

2. Name of the resulting member in LIB is member-name2 if parameter NEW is used, otherwise it is member-name-1

3. The language type is the one stored in the members identification record. The FORMAT parameter may be used to override the default format associated with the stored type or the one specified by parameter TYPE.

4. The UPDATE requests are ended with //EOD if no END parameter is specified; otherwise, by 'string' appearing in column 1 of a request record.

5. The command may be parameterized.

Requests:

1. Requests are interpreted according to the FORMAT parameter or to default values derived from TYPE or the unit type stored in its identification.

2. Requests and member-name-1 must be sorted on their line numbers in non-descending order.

3. If no request is given, the only effect of the command is to replace the language type stored in the unit identification with the one given by TYPE.

4. An update line whose number matches one or several lines in member-name-1 replaces the matched lines. Several update lines with equal line numbers are merged in sequence.

5. An update line with a line number that does not match a line number in member-name-1 is added to the source member in sequence (after the inserted lines if any).

6. An update line with string "$:D" in the first three positions of the text field causes the matched line(s) to be deleted. A range of lines may be deleted by placing the number of the last line to be deleted after the "$:D" . Intervening spaces are allowed; nothing else can appear on the update line.

7. An update line with line number field blank is assumed to have the same number as the previous update line. The line will be given number zero in the updated unit.

Examples:

- UPDATE UNIT1, NEW = UNIT2, REPLACE, NUMBER, TYPE = COBOL ;

```
␣␣␣230 REPLACED LINE
␣␣␣241 INSERTED LINE
␣␣␣250 $:D
␣␣␣310 $:D420
//EOD
```

UNIT1 is found by search rules. The resulting unit in LIB will be named UNIT2 and renumbered starting at 10 with increment 10.

Line 230 of UNIT1 is replaced, line 241 is inserted after line 240 of UNIT1, line 250 and lines 310 to 420 are deleted.

- UPDATE UNIT1, FORMAT = (1,2, 3, *), END = 'FIN' , CONTCHAR = '#' ;

```
25 FIRST PART OF REPLACED LINE #
WHICH CONTINUES ON #
```

```
THE FOLLOWING UPDATE #
RECORDS
35 $:D
FIN
```

Line number 25 is replaced, line 35 is deleted.

Function:

Convert source language library unit(s) into upper case letters.

Format:

```
        [(INLIB1 :)] [ ] (star-convention                               )
        [(INLIB2 :)] [ ] (Member-name   [, NEW = member-name-2]          )
UPPER   [(INLIB3 :)] [¬] (limited-star-convention-1                     )
        [(        )] [ ] ([, NEW = limited-star-convention-2]           )
        [(LIB :   )] [ ] ( (explicit-name-list)                          )
                         ( <indirect-name-list>                          )

                  [, REPLACE ];
```

Parameters:

-NEW Specifies the new name(s) to be given to the converted unit(s).

- FROM, TO may be used with star-convention or limited-star-convention.

Keywords :

-REPLACE Allow the unit(s) being written in output library LIB to replace (overwrite) an existing unit of the same name.

- if the specified or selected library is LIB, and if NEW is not specified, REPLACE need not be stated.

- REPLACE may be used even if no member with the same name exists in LIB.

Rules:

1. When member-name is used, search rules apply unless a library keyword is specified

2. When star-convention, limited-star-convention, explicit or indirect name list is used, default library is LIB unless a library keyword is specified

3. The command may be parameterized.

Examples:

UPPER LIB : MEMBER :

UPPER INLIB3 : <MYLIST> :

SECTION IX

COMMANDS APPLICABLE TO CU LIBRARIES

The following is a description of the commands that are applicable to Compile-Unit (CU) libraries.

A Compile-Unit library contains a number of members, each one being the result of a compilation. A CU has a name, which is generally the name of the compiled program or procedure. It may have alternate names to denote secondary entry points or, more generally, any catalogued external symbol definition. These alternate names are known as "aliases". An alias may be used in lieu of the Compile-Unit name and refers to the same entity.

Compile units have an identification in their own right. This will be displayed each time that a unit is handled by LIBMAINT.

Commands are presented in alphabetical order and each description starts at the top of a page.

DEL

Fun

Del

For

DEL

Par

- F

Rul

Exa

MOVE

Function:

- Move one or more CUs from COMFILE to a CU library.
- Move one or more CUs from a library to another CU sequential file.
- Move one or more CUs from a sequential file to a C

Format:

```
                                                  { member-name
                                                  { star-conventi
MOVE  [{INLIB1 :}]  [¬]                           {(explicit-name
      [{INLIB2 :}]
      [{INLIB3 :}]
      [{LIB :   }]
      [{COMFILE :}]
      [{INFILE :}]

                          [,END = 'string'] [,REPLA
```

- FROM, TO may be used with star-convention.
- END Denotes the end marker for input cards in COMFI
  - Maximum length of string is 8 characters.
  - All characters except single quotation mark (allowed.
  - If END is omitted string '//EOD' is assumed.
  - When COMFILE is used, string or //EOD is mand must appear in column 1 of the record followi input card.

Keywords:

- REPLACE Allow the unit being written in output librar replace (overwrite) an existing unit with th
  - When output is to OUTFILE, this keyword is the command will operate regardless of whe permission is given or not.
  - REPLACE may be used even if no member with name exists in LIB.

Rules:

1. When member-name is used, search rules apply, unless a library keyword, COMFILE or INFILE is specified.

2. When star-convention or explicit name list is used, default library is LIB unless a library keyword, COMFILE or INFILE is specified.

3. Library keyword LIB cannot be specified when LIBMAINT assigned output is OUTFILE.

4. When COMFILE is used, only member-name or * may be used (i.e. select one or all units in the following deck of cards).

5. When COMFILE is used, each CU entered on cards begins with a CU header holding the name of the unit and ends with an 'ECU' card. This is the format of the deck which would be produced by the PUNCH command.

6. The command may be parameterized.

- MOVE COMFILE : *, END = 'FIN' ;

```
{ card for CU1
ECU
{ cards for CU2
ECU
 .
 .
 .
FIN
```

Store CUs read from cards into LIB.

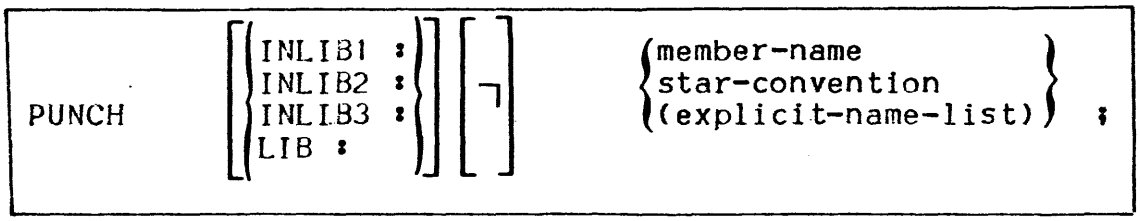
- MOVE INLIB2 : ?HB*, TO = HCOBOL, REPLACE ;

- MOVE INFILE : UNIT1 ;

Function:

Punch one or more CU(s) of a CU library.

Format:



Parameters:

- FROM, TO may be used with star-convention.

Rules:

1. When member-name is specified, search rules apply unless a library keyword is stated.

2. When star-convention or explicit name list is used, default library is LIB unless a library keyword is specified.

3. The command may be parameterized.

```
PUNCH INLIB2 : ALPHA ;

PUNCH LIB : *, FROM = ALPHA, TO = OMEGA ;
```

SECTION X

COMMANDS APPLICABLE TO LM LIBRARIES

The following is a description of the commands that are applicable to Load Module (LM) libraries.

A Load-Module library contains a number of members, each member being the result of the linkage of one or more Compile Units (CU) to produce an executable module acceptable by the GCOS-64 loader.

Load modules have an identification in their own right. This will be displayed each time that a unit is handled by LIBMAINT.

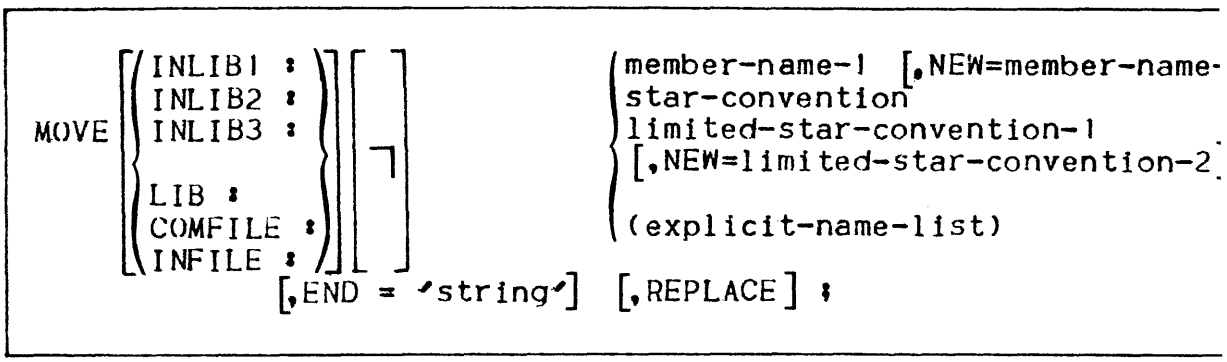
Commands are presented in alphabetical order and each description starts at the top of a page.

MOVE

Function:

- Move one or more LMs from the command input stream to a LI library.
- Move one or more LMs from a LM library to another LM libr sequential file.
- Move one or more LMs from a sequential file to a LM libra

Format:



Parameters:

- NEW — Specifies the name(s) of the resulting member(s); th may be used with member-name or limited-star-conven It may not be used when keyword COMFILE is present.
- FROM, TO — May be used with star-convention and limited-star-convention.
- END — Denotes the end of a sequence of input cards on COMF
  - maximum length of string is 8 characters.
  - all characters except single quotation mark(') are allowed.
  - if END is omitted, implicit value for 'string' is '//EOD'.
  - when COMFILE is specified, either //EOD or string mandatory and must appear in column one of the rec immediately following the last input card.

Keywords:

- REPLACE — Allow the member being written onto LIB to replace (overwrite) an existing member with the same name.
  - When LIBMAINT assigned output is OUTFILE, REPLACE ignored; the command operates regardless of whethe REPLACE permission is given or not.

- REPLACE may be used even if the resulting member name does not exist in LIB.

Rules:

1. When member-name-1 is specified, search rules apply unless a library keyword, INFILE or COMFILE is stated.

2. When star-convention or explicit name list is used, default library is LIB unless a library keyword, COMFILE or INFILE is specified.

3. Library keyword LIB cannot be specified when LIBMAINT assigned output is OUTFILE.

4. When COMFILE is used, only member-name-1 or * may be used (i.e.: select one or all units in the following deck of cards).

5. When COMFILE is used, each LM entered on cards begins with a LM header holding the name of the unit and ends with an 'ELM' card. This is the format of the deck which would be produced by the PUNCH command.

6. The command may be parameterized.

Examples:

- MOVE COMFILE : *, END = 'FIN' ;

```
  |
  |     cards for LM1
  |
ELM
  |
  |     cards for LM2
  |
ELM
.
.
.
FIN
```

  Store LMs read from cards into LIB.

- MOVE INLIB2 : PROC*, NEW = P*NEW, REPLACE ;

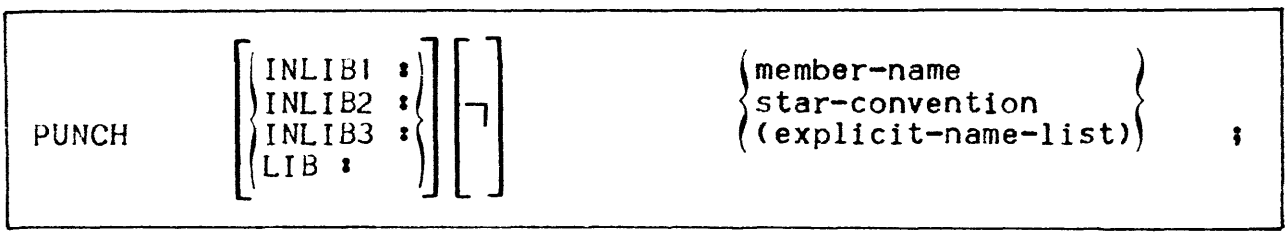
- MOVE INFILE : *, FROM = ALPHA, TO = OMEGA, REPLACE ;

PUNCH

Function:

Punch one or more LM(s) of a LM library.

Format:



Parameters:

- FROM, TO may be used with star-convention.

Rules:

1. When member-name is specified, search-rules apply unless a library keyword is used.
2. When star-convention or explicit name list is used, default library is LIB unless a library keyword is specified.
3. The command may be parameterized.

Examples:

```
PUNCH INLIB2 : *NEW ;

PUNCH PROG*3, FROM = PROGX3 ;
```

Function:

Change the name of one or more LM(s) in LIB.

Format:

```
RENAME [LIB:] { member-name-1,[NEW =] member-name-2                    } ;
              {[¬]limited-star-convention-1,                           }
              { [NEW =] limited-star-convention-2                      }
```

Parameters:

- NEW introduces the new name(s) of the renamed member(s).
- FROM, TO may be used with limited-star-convention.

Rules:

1. New member name(s) must not exist in LIB ; REPLACE parameter is not allowed.
2. Renamed members must exist in LIB.
3. The command may be parameterized.

Examples:

```
RENAME LIB : PLM*, NEW = QLM*NEW ;

RENAME *, *_NEW ;
```

SECTION XI

COMMANDS APPLICABLE TO SM LIBRARIES

The following is a description of commands that are applicable to sharable modules (SM) libraries. A SM library contains a number of members, each one being a SM. A SM contains in turn a number of members, each one being a Linked Unit (LKU).

The name of each member must be unique and must not exceed 30 characters in length.

A SM is the result of the linkage of LKU or of a LIBMAINT command. A LKU is the result of the linkage of one or more compile units that specify the containing SM.

Each description starts at the top of a page.

DELETE

Function:

To delete a SM and its associated LKUs in SM library LIB.

Format:

```
DELETE    SM = sm-name   ;
```

Example:

```
DELETE SM = MYSM ;
```

Function:

To initialize a SM in SM library LIB.

Format:

INIT SM = smname, STN = {E / F}, ESSTE = hexa-2 [,REPLACE] ;

Parameters:

- SM gives the name of the SM to be initialized.
- STN indicates the STN associated with the SM.
- ESSTE gives the Entry Segment STE ; this is an hexadecimal value limited to FF.

Keywords:

- REPLACE is needed if an existing SM with the same name already exists in LIB.

Example:

```
INIT SM = MYSM, STN = F, ESSTE = 09, REPLACE ;
```

LIST

Function:

To list part or all of a SM LIB or INFILE contents.

Format:

```
                                                    [,DETAILED]
LIST[INFILE,]  { DIR                                            }
               {  *                                             }
               {                                                }
               { SM = {sm-name}                                 }
               {      {   *   }                                 }
               {LKU = lku-name                                  }
                                                                 ;
```

Parameters and keywords:

- INFILE Specifies that the INFILE contents are to be listed. If not present, the LIB contents are listed.

- DIR or * To list the SMs and their associated LKU names.

- SM List the characteristics of a specified (sm-name) or all (*) SM s.

- DETAILED With SM = *, the associated LKU list is printed ; with SM = sm-name, the associated LKU name list and the entry points are listed (this latter combination is not allowed with INFILE).

- LKU Specifies the name of the LKU to be listed. Listed information includes segment number of private data and entry points (not allowed with INFILE).

Examples:

- LIST * ;
- LIST DIR ;
- LIST INFILE, * ;
- LIST INFILE, SM = MYSM ;
- LIST INFILE, SM = *, DETAILED ;
- LIST SM = *, DETAILED ;
- LIST LKU = MYLKU ;
- LIST SM = MYSM, DETAILED ;

Function:

To load one or all SMs from SM library LIB into backing store.

Format:

```
LOAD   SM = { sm-name } [,DEBUG]   [, REPLACE];
            {   *     }
```

Parameters and Keywords:

- SM — If SM = * all SMs are loaded ; otherwise the name of the SM to be loaded is given.

- DEBUG — Specifies that the SM is to be accessed by the Program Checkout Facility (PCF) during the execution of a LM referencing the SM. A SM loaded with option DEBUG may not be shared between process groups. To suppress the DEBUG option, the SM has to be CANCELled and reLOADed without this option.

- REPLACE — Is needed if a SM with the same name exists, after having been loaded from the same library.

Rules:

It is not allowed to load a SM from TEMP. SMLIB

Examples :

```
LOAD SM = MYSM, DEBUG ;
LOAD SM = * ;
LOAD SM = *, DEBUG ;
LOAD XM = MYSM, REPLACE, DEBUG ;
```

MOVE

Function:

To move one or all SMs of SM INLIB! or INFILE onto LIB or OUTFILE.

Format:

```
MOVE [INFILE,]   SM = { sm-name }      [,REPLACE] ;
                      {    *    }
```

Parameters and Keywords:

- INFILE if specified, SMs are moved from INFILE; if not specified, they are moved from INLIB!.
- SM states the name of the SM to be moved. If * is specified, all SMs will be moved.
- REPLACE is needed to overwrite an existing SM with the same name in LIB. This keyword is not allowed if LIBMAINT assigned output is OUTFILE.

Examples:

- MOVE INFILE, SM = * ,REPLACE ; (INFILE to LIB)
- MOVE SM = * ; (INLIB to LIB or OUTFILE)
- MOVE SM = MYSM, REPLACE ; (INLIB! to LIB)
- MOVE INFILE, SM = *; (INFILE to LIB or OUTFILE)

Function:

To clear the backing store of all information associated with a specified SM.

Format:

```
UNLOAD        SM = sm-name,  SMLIB = smlib-name ;
```

Parameters:

- SM gives the name of the SM as specified when the SM was initialized.

- SMLIB states the name of the SM library from which the SM has been loaded into backing store.

Example:

```
UNLOAD SM = MYSM SMLIB = MYLIB ;
```

SECTION XII

THE TEXT EDITOR

The TEXT EDITOR can be used to create, modify or edit an EBCDIC source unit in SSF format.

USAGE

The layout of the input to a text editor session is as follows :

```
EDIT-command
-
-
specific edit requests
-
-
```

Once the EDIT command is activated, all following records in the input stream are treated as editor requests until a Q (quit) is encountered. Requests fall into two general categories : input requests and edit requests. Input requests place the editor into input mode which allows the following EBCDIC records to be entered as new text until an appropriate escape sequence is read to switch the editor back into edit mode. Edit requests allow the user to read and write source units and to perform various simple or complex editing functions on their contents. Input and editing operations are not performed directly on the target units, but in a temporary buffer known as "Workspace".

REQUESTS

In the following list, the editor requests are divided into three categories : input requests, basic edit requests and extended edit requests. The basic edit requests are sufficient to allow a user to create and edit EBCDIC units and provide a good functional capability. Extended requests may be more time consuming but allow additional capability.

Input Requests

A - (append) : Enter input mode, append the following lines after specified line until the escape sequence is read.

C - (change) : Enter input mode, replace the specified line or lines with the following lines until the escape sequence is read.

I - (insert) : Enter input mode, insert the following lines before a specified line until the escape sequence is read.

Basic Edit Requests

D - (delete) : Delete specified line or lines from the workspace.

P - (print) : Print specified line or lines.

L - (print with line number) : Print specified line or lines prefixed with their internal line numbers.

Q - (quit) : exit from the text editor.

R - (read) : Read specified unit into workspace.

S - (substitute) : Replace specific character strings in specified line or lines.

W - (write) : Create a new unit with the contents of the edited workspace.

Z - (forced write) : Create a new unit with the contents of the edited workspace.

N - (no operation) : Do nothing.

- (count lines) : Count the number of lines with the specified content and print the result.

Extended Edit Requests

= - (print line number) : Print the line number (contained in the SSF header) of specified line.

G - (global) : Print, delete, print with the line number or print line number of all addressed lines that contain a specific character string.

V - (exclude) : Print, delete, print with the line number or print line number of all addressed lines that do not contain a

specific character string.

K - (copy) : Copy specified line or lines into a specified auxiliary workspace.

M - (move) : Move specified line or lines into a specified auxiliary workspace.

X - (status) : Print a summary status of all workspaces currently used.

T - (top of page) : Skip to top of page in the TEXT EDITOR output report.

F - (file output) : Alter the report output to the specified workspace.

E - (end file output) : Revert the report output to the normal default device.

O - (output message) : Print the remainder of the line on the output report.

: - (define label) : Set a label on this line for reference in a goto (>) request.

Y - (listing control) : Allow variations in the contents of the editors report.

% - (split line) : Split all lines containing a given pattern.

& - (concatenate lines) : Concatenate all lines containing a given pattern with the immediately preceding ones.

< - (backwards search) : Search for target line backwards.

> - (goto) : Skip n request lines or goto n request lines backwards or goto specified label.

* - (test contents) : If line contains the given string then execute the remainder of the request line, otherwise skip to next request line.

? - (range test) : If current line is in specified range then execute the remainder of the request line, otherwise skip to next request line.

ADDRESSING

The TEXT EDITOR is basically a line-oriented editor in that editing requests usually operate on an integral number of lines. As a result, most editing requests are preceded with an address specifying the line or lines in the workspace on which the request

is to operate. There are three basic means by which lines in the workspace can be addressed.

- addressing by line number
- addressing relative to the "current line"
- addressing by context

In addition, an address can be formed using a combination of the above techniques.

Addressing by Line Number

Each line in the workspace can be addressed by a decimal number indicating its line number as contained in its SSF header. Search for the target line is made sequentially from the "current line" to the last line of the workspace, then from the first line of the workspace to the one immediately preceding the "current line". Lines which are added to the workspace are given line number 0. Lines which are modified keep their original numbering.

Addressing Relative to the Current Line

The editor maintains the notion of a "current line" that is addressable by using the character "." (period) to represent the address of the current line. Normally, the current line is the last line addressed by an edit request or the last line entered by an input request. The value of "." after each editor request is documented in the description of the request and in the summary of requests at the end of this section .

Lines can be addressed relative to the current line number by using an address consisting of "." followed by a signed decimal number specifying the position of the desired line relative to the current line. For example the address ".+1" specifies the line immediately following the current line and the address ".-1" specifies the line immediately preceding the current line.

When specifying an increment to the current line position, the "+" sign can be omitted (e.g. ".5" is interpreted as ".+5"). In addition, when specifying a decrement to the current line position, the "." itself can be omitted (e.g. "-3" is interpreted as ".-3).
Two symbols are used to denote respectively the first and last line of a workspace.

- "⌝" denotes the first line ; hence "⌝+1" would denote the second line and "⌝+i" the (i+1)th. line of the workspace.

- "$"denotes the last line of the workspace, hence "$-1" would address the last but one line, etc...

Addressing by context

Lines can be addressed by context by using a regular expression to match a string of characters on a line. When used as an address, a regular expression specifies the first line encountered that contains a string of characters that matches the regular expression.

For example, in the following text the regular expression "/ABC/" matches line 2.

```
A : PROCEDURE ;
ABC : DEF ;
X = Y ;
END A ;
```

To use a regular expression as an address, the user enters "/regexp/", where "regexp" is any valid expression as described below. The search for a regular expression begins on the line following the current line (i.e. .+1) and continues through the entire workspace, if necessary, until it again reaches the current line. In other words the search proceeds from ".+1" to "$" and then from "¬" to ".". If the search is successful, "/regexp/" specifies the first line encountered during the search in which a match was found.

A regular expression can consist of any character in the EBCDIC set. However the following characters have specialised meaning in regular expressions.

"/" delimits a regular expression used as an address;

"*" signifies "any number (or none) of the preceding character" ;

"¬" when used as the first character of a regular expression, the "¬" character signifies the virtual character preceding the first character on a line ;

"$" when used as the last character of a regular expression, the "$" character signifies the virtual character following the last character on a line ;

"." matches any character on a line.

Some examples follow

/A/	Matche
/ABC/	Matche
/AB*C/ line	Matche
/IN..TO/	Matche charac
/IN.*TO/	Matche
/¬ABC/	Matche
/ABC$/	Matche
/¬ABC.*DEF$/	Matche
/.*/	Matche

The special meaning o expression can be nul "¢C". Thus :

/¢C/¢C*/	Matche
/¢C¬/	Matche

The editor remembers context. The user can using a null regular expression can be fol manner as when addres the addresses "/ABC/+ second line following

Note that the two use and (2) as special ch distinguished by cont

Compound Addresses

An address can b described above. guide in the for

- if a line numb first componen

specifies a series of two lines, th current line through the second lin

However, if a semi-colon is used to of a comma, the value of "." is set addressed by al before the evaluati to the example given immediately ab

.1;.2

specifies a series of three lines, following the original current line following the line specified by al. address pair :

/ABC/;.+1

is equivalent to the address pair :

/ABC/,/ABC/

Addressing Errors

The following list describes the various the editor is attempting to evaluate an

- "WORKSPACE EMPTY" - an attempt has specific line when the workspace is are legal addresses within an empty with a read, append or insert reque
- "ADDRESS NEGATIVE" or "ADDRESS TOO made to refer to a non-existent lin when there are fewer than 20 lines address of ¬1 - 4).
- "ADDRESS WRAP AROUND" - an attempt series of lines in which the positi addresses is before the line addres (e.g. $,¬).
- "SEARCH FAILED" - a regular expressi search initiated from the request s matching line.
- "SYNTAX ERROR IN REG-EXP" - a regul address has not been properly delimi
- "//UNDEFINED" - a null regular expre previously defined regular expressi

Request Format

A request to the editor can take any one of the following forms depending on the number of addresses to be specified with the request :

```
<request>

adr <request>

adr1,adr2<request>

adr1;adr2<request>
```

adr, adr1 and adr2 are any legal address as specified above, and request is any valid editor request.

Some editor requests require no address, some require a single address and others require a pair of addresses. In all cases, however, the user can use a request omitting one or both of the required addresses and let the editor provide the missing address information by default. The following rules apply to the use of addresses specified by default

- if a request requiring an address pair is issued with the second address missing, the (missing) second address is assumed to be the same as the first. For example:

  ```
  adr<request>
  ```

  is interpreted as:

  ```
  adr,adr<request>
  ```

  and addresses a single line in the workspace (i.e. the line addressed by adr)

- if a request requiring an address pair is issued with both addresses missing, one of the following address pairs is assumed depending on the request issued:

  ```
  .,. <request> for most editor requests

  1,$ <request> for write, forced write, global and exclude
  ```

- if a request requiring a single address is issued with no address specified, one of the following addresses is assumed depending on the request issued

  ```
  . <request> for most editor requests

  $ <request> for read requests
  ```

The Value of "."

All editor requests that alter the contents of the workspace or cause information to be output change the value of "." (i.e. the current line). Usually, the value of "." is set to the last line address specified (either explicitly or by default) in the editor request. The one major exception to this rule is the delete request which sets "." to the line after the last line deleted.

Multiple Requests on a Line

In general, any number of editor requests can be issued in a single input line. However, each of the requests listed below must terminate a line and, thus, must appear on a line by itself or be placed at the end of a line containing multiple editor requests.

R read
W and Z write and forced write
A,C and I input requests
Q quit request

Spacing

The following rules govern the use of spaces in editor requests:

- spaces are taken as literal when appearing inside regular expressions: thus, /THE N/ is not the same as /THEN/
- spaces cannot appear in numbers, i.e. 13 cannot be written as 1 3 (which is interpreted as 1+3)
- spaces within addresses, except as indicated above, are ignored
- the treatment of spaces in the body of an editor request depends on the nature of the request

Comments

The quotation mark character (") is reserved as the comment delimiter and is actually implemented as an editor request, the effect of which is to ignore the remainder of the request line. If the quotation mark is preceded by an address, the value of

"." is set to that address.

The Locate Request

If an address terminates a request line, the value of "." is set to the addressed line and the line is printed. For example the request line:

```
/¬START/
```

locates a line beginning with START, sets the value of "." to that line and prints it.

Responses from the Editor

In general, the editor does not respond with output unless explicitly requested to do so (e.g. with a print line number request).

The use of frequent print requests is recommended for users using the editor for the first time.

If an error is encountered by the editor, an error message is printed and a skip to the next request line is made. Thus the trailing part of the offending request line is ignored.

INPUT MODE

The editor can be placed in input mode with the use of one of the three input requests (append, change and insert). The input request must terminate a request line. It is followed by a number of literal text lines.

The literal text can contain any number of EBCDIC source lines. To exit from input mode and terminate the input request, the escape sequence ¢F is entered as the first character of a new line. The usual form of an input request is as follows:

```
adr1 [,adr2]<input request>
-
-
text
-
-
 ¢F
```

It is important to remember to terminate the input request with the ¢F escape before entering another request. Otherwise the (would be) editor request is regarded as input and included in the text rather than executed as a request.

Upon leaving input mode, the value of "." is set to the last input line. The special meaning of any of the escape sequences used by the editor (e.g.¢F,¢C,¢B and ¢X) can be suppressed by inserting the ¢C escape sequence betwen the two characters (e.g.¢¢CF,¢¢CC,¢¢CE.¢¢CX), thus allowing the escape sequence to be input as literal text. All input lines entered are assigned line number zero and should therefore be later addressed by context or relative addressing rather than by line number.

specific character string.

K - (copy) : Copy specified line or lines into a specified auxiliary workspace.

M - (move) : Move specified line or lines into a specified auxiliary workspace.

X - (status) : Print a summary status of all workspaces currently used.

T - (top of page) : Skip to top of page in the TEXT EDITOR output report.

F - (file output) : Alter the report output to the specified workspace.

E - (end file output) : Revert the report output to the normal default device.

O - (output message) : Print the remainder of the line on the output report.

: - (define label) : Set a label on this line for reference in a goto (>) request.

Y - (listing control) : Allow variations in the contents of the editors report.

% - (split line) : Split all lines containing a given pattern.

& - (concatenate lines) : Concatenate all lines containing a given pattern with the immediately preceding ones.

< - (backwards search) : Search for target line backwards.

> - (goto) : Skip n request lines or goto n request lines backwards or goto specified label.

* - (test contents) : If line contains the given string then execute the remainder of the request line, otherwise skip to next request line.

? - (range test) : If current line is in specified range then execute the remainder of the request line, otherwise skip to next request line.

ADDRESSING

The TEXT EDITOR is basically a line-oriented editor in that editing requests usually operate on an integral number of lines. As a result, most editing requests are preceded with an address specifying the line or lines in the workspace on which the request

is to operate. There are three basic means by which lines in the workspace can be addressed.

- addressing by line number
- addressing relative to the "current line"
- addressing by context

In addition, an address can be formed using a combination of the above techniques.

Addressing by Line Number

Each line in the workspace can be addressed by a decimal number indicating its line number as contained in its SSF header. Search for the target line is made sequentially from the "current line" to the last line of the workspace, then from the first line of the workspace to the one immediately preceding the "current line". Lines which are added to the workspace are given line number 0. Lines which are modified keep their original numbering.

Addressing Relative to the Current Line

The editor maintains the notion of a "current line" that is addressable by using the character "." (period) to represent the address of the current line. Normally, the current line is the last line addressed by an edit request or the last line entered by an input request. The value of "." after each editor request is documented in the description of the request and in the summary of requests at the end of this section .

Lines can be addressed relative to the current line number by using an address consisting of "." followed by a signed decimal number specifying the position of the desired line relative to the current line. For example the address ".+1" specifies the line immediately following the current line and the address ".-1" specifies the line immediately preceding the current line.

When specifying an increment to the current line position, the "+" sign can be omitted (e.g. ".5" is interpreted as ".+5"). In addition, when specifying a decrement to the current line position, the "." itself can be omitted (e.g. "-3" is interpreted as ".-3).
Two symbols are used to denote respectively the first and last line of a workspace.

- "⌝" denotes the first line ; hence "⌝+1" would denote the second line and "⌝+i" the (i+1)th. line of the workspace.

- "$"denotes the last line of the workspace, hence "$-1" would address the last but one line, etc...

Addressing by context

Lines can be addressed by context by using a regular expression to match a string of characters on a line. When used as an address, a regular expression specifies the first line encountered that contains a string of characters that matches the regular expression.

For example, in the following text the regular expression "/ABC/" matches line 2.

```
A : PROCEDURE ;
ABC : DEF ;
X = Y ;
END A ;
```

To use a regular expression as an address, the user enters "/regexp/", where "regexp" is any valid expression as described below. The search for a regular expression begins on the line following the current line (i.e. .+1) and continues through the entire workspace, if necessary, until it again reaches the current line. In other words the search proceeds from ".+1" to "$" and then from "¬" to ".". If the search is successful, "/regexp/" specifies the first line encountered during the search in which a match was found.

A regular expression can consist of any character in the EBCDIC set. However the following characters have specialised meaning in regular expressions.

"/" delimits a regular expression used as an address;

"*" signifies "any number (or none) of the preceding character" ;

"¬" when used as the first character of a regular expression, the "¬" character signifies the virtual character preceding the first character on a line ;

"$" when used as the last character of a regular expression, the "$" character signifies the virtual character following the last character on a line ;

"." matches any character on a line.

Some examples follow :

/A/	Matches the letter A anywhere on a line
/ABC/	Matches the string ABC anywhere on a line
/AB*C/	Matches AC, ABC, ABBC, ABBBC, etc... anywhere on a line
/IN..TO/	Matches a line containing IN followed by any two characters followed by TO
/IN.*TO/	Matches a line containing IN and TO in that order
/¬ABC/	Matches a line beginning with ABC
/ABC$/	Matches a line ending with ABC
/¬ABC.*DEF$/	Matches a line beginning with ABC and ending with DEF
/.*/	Matches any line

The special meaning of "/", "*", "$", "¬" and "." within a regular expression can be nullified by preceding the special character with "¢C". Thus :

/¢C/¢C*/	Matches the string /* anywhere on a line
/¢C¬/	Matches any line containing the character ¬

The editor remembers the last regular expression used in any context. The user can reinvoke the last used regular expression by using a null regular expression (i.e"//"). In addition, a regular expression can be followed by a signed decimal integer in the same manner as when addressing relative to the current line. For example, the addresses "/ABC/+5-3", "/ABC/+2" or "/ABC/2" all address the second line following a line containing ABC.

Note that the two uses of ".", "¬" and "$" (1) as line addresses and (2) as special characters in regular expressions) are distinguished by context.

Compound Addresses

An address can be formed using a combination of the techniques described above. The following rules are intended as a general guide in the formation of these compound addresses.

- if a line number is to appear in an address, it must be the first component of the address.

- a line number can be followed by a regular expression. This construct is used to begin the regular expression search after a specific line number. For example, the address "10/ABC/" starts the search for "/ABC/" immediately after line number 10.

- a regular expression can follow an address specified relative to the current line number. For example, the address ".-8/ABC/" starts the search from 8 lines preceding the current line.

- a regular expression can be followed by another regular expression. For example, the address "/ABC//DEF/" matches the first line containing DEF appearing after the first line containing ABC. As mentioned earlier, a regular expression can be followed by a decimal integer. For example, the address "/ABC/-10/DEF/5" starts the search for "/DEF/" from 10 lines preceding the first line to match "/ABC/" and, if "/DEF/" is matched, the value of the compound address is the fifth line following the line containing the match for "/DEF/".

Addressing a Series of Lines

Several of the editor requests can be used to operate on a series of lines in the workspace. To specify a series of lines, two addresses must be given in the following general form.

.a1,a2

The pair of addresses specifies the series of lines starting with the line addressed by the address a1 through the line addressed a2 inclusive.

Examples :

1,5	specifies from line number 1 through line number 5
⁊, $	specifies the entire contents of the workspace
.1,/ABC/	specifies the line following the current line through the first line after the current line containing ABC

When a comma is used to separate addresses, the address computation of the second address is unaffected by the computation of the first address (i.e. the value of "." is not changed by the evaluation of the first address). For example, the address pair :

.1,.2

specifies a series of two lines, the line immediately after the current line through the second line after the current line.

However, if a semi-colon is used to separate addresses instead of a comma, the value of "." is set to point to the line addressed by a1 before the evaluation of a2 begins. In contrast to the example given immediately above, the address pair :

```
.1;.2
```

specifies a series of three lines, the line immediately following the original current line through the second line following the line specified by a1. As a further example, the address pair :

```
/ABC/;.+10
```

is equivalent to the address pair :

```
/ABC/,/ABC/+10
```

Addressing Errors

The following list describes the various errors that can occur when the editor is attempting to evaluate an address.

- "WORKSPACE EMPTY" - an attempt has been made to reference a specific line when the workspace is empty. (Only "$" and "¬" are legal addresses within an empty workspace and only if used with a read, append or insert request.)

- "ADDRESS NEGATIVE" or "ADDRESS TOO BIG" - an attempt has been made to refer to a non-existent line (e.g. an address of ¬ + 20 when there are fewer than 20 lines in the workspace or an address of ¬ - 4).

- "ADDRESS WRAP AROUND" - an attempt has been made to address a series of lines in which the position of the second line addresses is before the line addressed by the first address (e.g. $,¬).

- "SEARCH FAILED" - a regular expression search or a line number search initiated from the request stream has failed to find a matching line.

- "SYNTAX ERROR IN REG-EXP" - a regular expression used as an address has not been properly delimited.

- "//UNDEFINED" - a null regular expression has been used and no previously defined regular expression is available.

Request Format

A request to the editor can take any one of the following forms depending on the number of addresses to be specified with the request :

```
<request>

adr <request>

adr1,adr2<request>

adr1;adr2<request>
```

adr, adr1 and adr2 are any legal address as specified above, and request is any valid editor request.

Some editor requests require no address, some require a single address and others require a pair of addresses. In all cases, however, the user can use a request omitting one or both of the required addresses and let the editor provide the missing address information by default. The following rules apply to the use of addresses specified by default

- if a request requiring an address pair is issued with the second address missing, the (missing) second address is assumed to be the same as the first. For example:

  ```
  adr<request>
  ```

  is interpreted as:

  ```
  adr,adr<request>
  ```

  and addresses a single line in the workspace (i.e. the line addressed by adr)

- if a request requiring an address pair is issued with both addresses missing, one of the following address pairs is assumed depending on the request issued:

  ```
  .,. <request> for most editor requests

  1,$ <request> for write, forced write, global and exclude
  ```

- if a request requiring a single address is issued with no address specified, one of the following addresses is assumed depending on the request issued

  ```
  . <request> for most editor requests

  $ <request> for read requests
  ```

The Value of "."

All editor requests that alter the contents of the workspace or cause information to be output change the value of "." (i.e. the current line). Usually, the value of "." is set to the last line address specified (either explicitly or by default) in the editor request. The one major exception to this rule is the delete request which sets "." to the line after the last line deleted.

Multiple Requests on a Line

In general, any number of editor requests can be issued in a single input line. However, each of the requests listed below must terminate a line and, thus, must appear on a line by itself or be placed at the end of a line containing multiple editor requests.

R read
W and Z write and forced write
A,C and I input requests
Q quit request

Spacing

The following rules govern the use of spaces in editor requests:

- spaces are taken as literal when appearing inside regular expressions: thus, /THE N/ is not the same as /THEN/
- spaces cannot appear in numbers, i.e. 13 cannot be written as 1 3 (which is interpreted as 1+3)
- spaces within addresses, except as indicated above, are ignored
- the treatment of spaces in the body of an editor request depends on the nature of the request

Comments

The quotation mark character (") is reserved as the comment delimiter and is actually implemented as an editor request, the effect of which is to ignore the remainder of the request line. If the quotation mark is preceded by an address, the value of

"." is set to that address.

The Locate Request

If an address terminates a request line, the value of "." is set to the addressed line and the line is printed. For example the request line:

```
/¬START/
```

locates a line beginning with START, sets the value of "." to that line and prints it.

Responses from the Editor

In general, the editor does not respond with output unless explicitly requested to do so (e.g. with a print line number request).

The use of frequent print requests is recommended for users using the editor for the first time.

If an error is encountered by the editor, an error message is printed and a skip to the next request line is made. Thus the trailing part of the offending request line is ignored.

INPUT MODE

The editor can be placed in input mode with the use of one of the three input requests (append, change and insert). The input request must terminate a request line. It is followed by a number of literal text lines.

The literal text can contain any number of EBCDIC source lines. To exit from input mode and terminate the input request, the escape sequence ¢F is entered as the first character of a new line. The usual form of an input request is as follows:

```
adr1 [,adr2]<input request>
-
-
text
-
-
 ¢F
```

It is important to remember to terminate the input request with the ¢F escape before entering another request. Otherwise the (would be) editor request is regarded as input and included in the text rather than executed as a request.

Upon leaving input mode, the value of "." is set to the last input line. The special meaning of any of the escape sequences used by the editor (e.g.¢F,¢C,¢B and ¢X) can be suppressed by inserting the ¢C escape sequence betwen the two characters (e.g.¢¢CF,¢¢CC,¢¢CE.¢¢CX), thus allowing the escape sequence to be input as literal text. All input lines entered are assigned line number zero and should therefore be later addressed by context or relative addressing rather than by line number.

FUNCTION:

The append request is used to enter input lines from the input stream, appending these lines after the line addressesd by the append request. The append request is one of the few requests that can operate correctly when the workspace is empty.

FORMAT:

```
adrA
-
-
text
-
-
¢F
```

DEFAULT:

A is taken to mean .A

VALUE OF ".":

set to the last line appended

EXAMPLE:

```
- Before -              A : PROCEDURE ;
                        X = Y ;
                        END A ;

- request sentence -¬ + 1A
                        Q = R ;
                        c F

- After -               A : PROCEDURE ;
                        X = Y ;
                "." -> Q = R ;
                        END A ;
```

Note : request $A can be used to insert new text at the end of a workspace.

FUNCTION:

The change request is used to delete an addressed line or range of lines and replace the deleted line(s) with new text read from the input stream.

FORMAT:

```
adr1, adr2C
-
-
text
-
-
¢F
```

DEFAULT:

C is taken to mean .,. C

adC is taken to mean ad, adC

VALUE OF ".":

set to the last line entered

EXAMPLE:

- Before -

```
A : PROCEDURE ;
X = Y ;
Q = R ;
END A ;
```

- request sentence -

```
¬1,¬2C
S = T ;
U = V ;
W = Z ;
¢F
```

- After -

```
A : PROCEDURE ;
S = T ;
U = V ;
"." -> W = Z ;
END A ;
```

FUNCTION:

The insert request is used to enter input lines from the input stream and insert the new text immediately before the addressed line. The insert request is one of the few requests that can operate on an empty workspace.

FORMAT:

```
adr I
-
-
text
-
-
¢F
```

DEFAULT:

I is taken to mean .I

VALUE OF ".":

set to the last line inserted

EXAMPLE:

```
- Before -              A : PROCEDURE ;
                        X = Y ;
                        END A ;

- request sentence -/X = /I
                        Q = R ;
                        ¢F

- After -               A : PROCEDURE ;
          "." ->        Q = R ;
                        X = Y ;
                        END A ;
```

Note: request adrI has the same effect as the request adr-1A. Request ⅂I is used to insert text before the first line of the workspace.

The basic edit requests described below represent a subset of editor suitable for most editing situations. Additional requests are described later in this section under "Extended edit requests" and "Auxiliary workspaces".

Delete Request (D)

FUNCTION:

The delete request is used to delete the addressed line or set of lines from the workspaces.

FORMAT:

adr1, adr2D

DEFAULT:

D is taken to mean .,. D

adD is taken to mean ad,adD

VALUE OF ".":

set to the line immediately following the last line deleted

EXAMPLE:

```
- Before -          A : PROCEDURE ;
                    X = Y ;
                    Q = R ;
                    S = T ;
                    END A ;

- request sentence -/Q=/ , /S=/D

- After -           A : PROCEDURE ;
                    X = Y ;
          "." ->    END A ;
```

FUNCTION:

The print requests are used to print the addressed line or set of lines: P prints the addressed line(s) without line number;L prints the addressed line(s) and prefixes them with their internal line numbers.

FORMAT:

adr1, adr2P or adr1, adr2L

DEFAULT:

P or L are taken to mean .,.P or .,.L

adP or adL are taken to mean ad,adP or ad,adL

VALUE OF ".":

set to the last line addressed by the request(i.e. the last line to be printed)

EXAMPLE:

- contents of workspace -

```
A : PROCEDURE ;
X = Y ;
Q = R ;
S = T ;
END A ;
```

- request sentence -

```
/X=/,/S=/P
```

- printed output -

```
      X = Y ;
      Q = R ;
"." -> S = T ;
```

Quit Request (Q)

FUNCTION:

The quit request is used to exit from the editor and does not itself save the result of any editing that might have been done. If the user wishes to save the modified contents of the workspace, he must explicity use a write or forced write request (see below).

FORMAT:

Q

DEFAULT:

The quit request cannot have an address

NOTE: the quit request must terminate a request line; the remainder of the line is treated as a comment.

FUNCTION:

The read request is used to append the contents of a specified source unit after the addressed line. The read request is one of the few requests that can operate on an empty workspace.

FORMAT:

adrRname

name is the name of a SSF source unit in a library to be read in the workspace after the line addressed by adr. The name of the unit follows the syntactical rules for a unit name in LIBMAINT (i.e. :[lib:] name) ; it can be preceded by any number of spaces and must terminate the request line.

DEFAULT:

Rname is taken to mean $Rname

VALUE OF ".":

set to the last line read from the unit

EXAMPLE:

```
- before -                    A : PROCEDURE ;
                              X = Y ;
                              END A ;

- request sentence -          /X=/R BX
```

where BX contains the following text

```
                              B : PROCEDURE ;
                              C = D ;
                              END B ;

- after -                     A : PROCEDURE ;
                              X = Y ;
                              B : PROCEDURE ;
                              C = D ;
                      "."-> END B ;
                              END A ;
```

Substitute Request (S)

FUNCTION:

The substitute request is used to modify the contents of the addressed line or set of lines by replacing all strings that match a given regular expression with a specified character string.

FORMAT:

adr1,adr2S/regexp/string/

(the first character after S is taken to be the regular expression delimiter and can be any character not appearing in either regexp or in string).

DEFAULT:

S/regexp/string/ is taken to mean .,.S/regexp/string/

adS/regexp/string is taken to mean ad, adS/regexp/string

VALUE OF ".":

set to the last line addressed by the request

OPERATION:

Each character string in the addressed line or lines that matches regexp is replaced with the character string. If string contains character &, each & is replaced by the string matched by regexp. The special meaning of & can be suppressed by preceding & with the escape sequence ¢C.

EXAMPLE:

```
- Before -          THE QUICK BROWN SOX
- Request           S/SOX/FOX/
- After -           THE QUICK BROWN FOX

- Before -          XYZINDEX = Q ;
- Request -         S?INDEX?(&)?
- After -           XYZ(INDEX) = Q;

- Before -          X = Y
- Request -         S/$/;/
- After -           X = Y ;
```

FUNCTION:

The write and forced write requests are used to write the addressed line or set of lines into a specific source unit. If the source unit already exists, request W will be rejected and request Z will result in the overwriting of the existing unit with the addressed line(s).

FORMAT:

adr1,adr2 {W|Z} [(type)] [LIB:] name

"name" is the name of a source unit to be created (W) or overwritten (Z) in the specified library. The name of the unit must not exceed 31 characters in length ; it can be preceded with any number of spaces and must terminate the request line.

"type" is the language type to be set in control records of the output library. Applicable values are the same as for parameter TYPE in the MOVE SL command.

DEFAULT:

Wname or Zname are taken to mean ,$Wname or ,$Zname.
adWname or adZname are taken to mean ad,adWname or ad,adZname.
If type is omitted, DAT is assumed unless the member already exists in LIB, in which case the existing type is preserved.

VALUE OF ".":

unchanged

EXAMPLES:

```
W(COB)MYPROGRAM
Z UNIT-A
Z(COB)M2
```

The No-operation Request (N)

FUNCTION:

The no-operation request N is used to position on a line without issuing any output on the report. It is identical to the locate request with the difference that the located line is not printed.

FORMAT:

adr N

adr is the address of the line to position on .

DEFAULT:

N is taken to mean .N

VALUE OF ".":

set to adr.

EXAMPLES:

```
.N
/PROCEDURE DIVISION/N
```

FUNCTION:

The count lines request counts the number of lines which, in a specified range, contain the given regular expression. The count is printed on the execution report.

FORMAT:

adr1,adr2#/regexp/

- adr1,adr2 specify the range of the counting (first line, last line)
- regexp specifies the regular expression

(the first character after # is taken to be the regular expression delimiter and can be any character not appearing in regexp)

DEFAULT:

#/regexp/is taken to mean .,.#/regexp/
ad#/regexp/ is taken to mean ad,ad#/regexp/

VALUE OF ".":

set to adr2

EXAMPLES:

```
1,$#/X=3/
130,500#/SECTION/
```

The editor requests discussed up to this point comprise a basic subset sufficient for most applications. A user learning to use the editor for the first time might be well advised to stop at this point.

Print Line Number Request (=)

FUNCTION:

This request is used to print the line number (as contained in the SSF header and not to be mistaken with the rank of the line in the workspace) of the addressed line.

FORMAT:

```
adr =
```

DEFAULT:

= is taken to mean .=

VALUE OF ".":

set to the line addressed by the request

EXAMPLE:

```
- contents of the workspace -

- SSF header -      - text-

1000               A : PROCEDURE ;
1100               X = Y ;
1300               P = Q ;
1800               END A ;

- request - ¬1+2= or /Q;/=

- response - 1300
```

FUNCTION:

The global request is used in conjunction with some other request (e.g. print, print with number, print line number, delete).
That request is to operate only on those lines addressed by the global request that contain a match for a specified regular expression.

FORMAT:

adr1,adr2Gx/regexp/
where "x" must be one of the following requests:

D delete lines containing "regexp";
P print lines containing "regexp";
L print with number lines containing "regexp";
= print the numbers of lines containing "regexp";

DEFAULT:

Gx/regexp/ is taken to mean ,$Gx/regexp/
adGx/regexp/ is taken to mean ad,ad Gx/regexp/

VALUE OF ".":

set to adr2 of request

NOTE:

The character immediately following the request x is taken to be the regular delimiter and can be any character not appearing in "regexp"

EXAMPLE:

```
- Before -                    A : PROCEDURE ;
                              Q = R ;
                              X = Y ;
                              END A ;

- Request -                  ¬],$GD/Q/

- After -                     A : PROCEDURE ;
                              X = Y ;
                    ".."->    END A ;
```

Exclude Request (V)

FUNCTION:

The Exclude request is used in conjunction with some other request (e.g. print, print with number, print line number, delete). That request is to operate only on those lines addressed by the exclude request that do not contain a match for a specified regular expression.

FORMAT:

adr1, adr2Vx/regexp/

where "x" must be one of the following requests:

D delete lines not containing "regexp" ;
P print lines not containing "regexp" ;
L print with number lines not containing "regexp" ;
= print the numbers of lines not containing "regexp"

DEFAULT:

Vx/regexp/ is taken to mean ,$Vx/regexp/
ad Vx/regexp/ is taken to mean ad, ad Vx/regexp/

VALUE OF ".":

set to adr2 of request

NOTE:

The character immediately following the request x is taken to be the regular expression delimiter and can be any character not appearing in "regexp".

EXAMPLE:

- Before -
```
A : PROCEDURE ;
  Q = R ;
  X = Y ;
  X = Q ;
  END A ;
```

- Request - ⌐],$VP/Q/

- Response -
```
A : PROCEDURE ;
  X = Y ;
  END A ;
```

AUXILIARY WORKSPACES

The discussion up to this point has assumed the existence of only one single workspace. Actually the editor supports up to 6 different workspaces. One workspace at a time can be designated as the "current workspace": any other workspaces at this time are referred to as "auxiliary workspaces". All the editor requests described so far operate within the current workspace.

Each workspace is given a symbolic name : "0","1","2","3","4" or "5". When the editor is invoked a single workspace (workspace "0") is activated and designated as the current workspace. Additional workspaces can be created merely by referencing a previously undefined workspace name.

Workspace names are usually enclosed between parentheses: however these may omitted (e.g., "5" is taken to be "(5)").

Change Workspace (B)

FUNCTION:

The change workspace request is used to designate an auxiliary workspace as the current workspace. The previously designated current workspace becomes an auxiliary workspace.

FORMAT:

B(x) or Bx

where "x" is the name of the workspace to become the current workspace.

VALUE OF ".":

restored to the value of "." when workspace "x" was last used as current workspace (i.e., the value of "."is maintained separately for each workspace and saved as part of the workspace status).

EXAMPLE:

```
B(5)
B4
```

Copy and Move Requests (K and M)

FUNCTION:

The copy and move requests are used to copy or move one or more lines to a specified auxiliary workspace. The addressed lines replace the previous contents (if any) of the auxiliary workspace.

FORMAT:

adr1,adr2{M}(x) or adr1,adr2 {M} x
{K} {K}

where "x" is the name of the auxiliary workspace to which lines are to be copied or moved

DEFAULT:

M(x) or K(x) are taken to mean .,.M(x) or .,.K(x)

adM(x) or adK(x) are taken to mean ad,adM(x) or ad,adK(x)

VALUE OF ".":

set to last copied line for K or the line after the last copied for M in the current workspace ; set to 0 in the specified auxiliary workspace

EXAMPLE:

```
    - Before -

    - Current workspace -    - Workspace 2 -

    A : PROCEDURE            ABC = DEF ;
    X = Y ;                  END BIN ;
    Y = K ;
    K = R ;
    END A ;

    -Request -          /K;/,/R;/M(2)

    - After -

    - Current Workspace -    - Workspace 2 -

    A : PROCEDURE            Y = K ;
    X = Y ;                  K = R ;
"."->END A ;
```

Request /K;/,/R;/K(2) would have left the current workspace unchanged and given the same contents for workspace 2.

FUNCTION:

The workspace status request is used to print a summary of the status of all workspaces currently in use. The name and length (in lines) of each workspace is listed : the current workspace is marked with a right arrow "--->" immediately to the right of the workspace name.

FORMAT:

X

VALUE OF ".":

unchanged

EXAMPLE:

If the user has created the additional workspaces 2 and 4 and has designated 2 as his current workspace, the output of the workspace status request might be as follows:

```
*WORKSPACE(0)       157
*WORKSPACE(2) --->32
*WORKSPACE(4)       53
```

This output indicates 157 lines in workspace 0(the initial workspace);32 lines in workspace 2(the current workspace) and 53 lines in workspace 4.

Input to the Editor can be viewed as a stream of EBCDIC lines. Depending on the context, some of these lines are interpreted as Editor requests and others are interpreted as literal text. The "¢B(x)" escape sequence is recognized by the Editor in either context as a directive to alter the input stream to read subsequent lines from workspace "x".

When the text Editor encounters the sequence "¢B(x)", the entire escape sequence is removed from the stream and replaced with the literal contents of the specified workspace. The text Editor proceeds exactly as if the the current content of workspace x were in the request stream in place of¢B(x). If another "¢B" escape sequence is encountered while accepting input from workspace "x"(i.e. appears in the literal contents of workspace x), the newly encountered escape sequence also is replaced by the contents of the named workspace. The text Editor allows the recursive replacement of "¢B" escape sequences by the contents of named workspaces to a depth of 50 nested¢B escape sequences.

Request Stream	Workspace X	Workspace Y
1	a	i
2	b	j
3	c	k
4	d	l
¢B(x)	e	m
5	¢B(y)	n
6	f	o
7	g	
8	h	

is equivalent to the series of lines

1 2 3 4 a b c d e i j k l m n o f g h 5 6 7 8

EXAMPLE OF USE OF ¢B

The workspace to which the input stream is redirected can contain Editor requests, literal text, or both.
If the Editor is executing a request obtained from a workspace (rather than from the command stream) and the request specified a line number or regular expression for which no match is found, the usual error comment is suppressed and the remaining contents of the workspace are skipped. The escape sequence "¢B(x)" can be thought of as a subroutine call statement, and the failure to match a line or regular expression specified by some request in workspace "x" can be thought of as a return statement.

NOTE : The special meaning of "¢B" can be suppressed by preceding the character B with a "¢C" escape sequence (¢¢CB).

Use of Workspace for Moving Text

Perhaps the most common use of workspaces in the editor is for moving text from one part of a unit to another. A typical pattern is to move the text to be moved into an auxiliary workspace with a Move request. For example the request:

```
18,32M(5)
```

moves lines from line number 18 through line number 32 inclusive into auxiliary workspace 5. Once the lines have been moved to an auxiliary workspace, they can be read as literal text in conjunction with an input request. For example, to insert the lines in workspace 5 immediately before the last line in the current workspace, the following sequence might be used:

```
$I
¢B(5)¢F
```

In this case, the literal text in workspace 5 replaces the ¢B escape sequence and thus is treated as input to the editor already put in input mode by the Insert (I) request. Notice that the ¢F immediately following the ¢B escape sequence is correct since it can be expected that the last line in workspace 5 is terminated by a fictitious end of line mark that precedes the ¢F after the ¢B(5) is expanded.

Other Uses of Workspaces

Another common use for workspaces is to define frequently used editing sequences. For example, to add the same source code sequence in several places in a program, the programmer might elect to enter the editing sequence into a workspace only once and invoke the contents of the workspace as many times as necessary.

The use of workspaces also allows a user to place more elaborate Text Editor request sequences into auxiliary workspaces and use the Editor as a pseudo-programming language. In this context, it is useful to regard a workspace containing executable Editor requests as a subroutine and to view the "¢B" escape sequence as a call statement.
The reader should refer to the "LIBRARY MAINTENANCE User Guide" for specific examples of use.

The File Output and End File Output Requests (F and E)

In normal mode, all results issued by the text editor are

printed on the execution report. This might be altered by means of the F request which forces results, with the exception of error messages, to be appended at the end of the specified workspace. Request E returns to the normal reporting device.

This might be used, for example, in the following circumstance:

example : Construct a unit that contains all lines containing the string "CALL" or "RETURN" in unit X1. The following sequence of requests:

```
R X1
F (1)
¬1,$GP/CALL/
¬1,$GP/RETURN/
E
```

achieves the requested objective. Instead of printing lines containing "CALL" and "RETURN", the editor appends them in workspace 1 which might later be used by other requests or written into the library.

The Output Message Request (O)

The O request causes the remainder of the request line to be printed on the output report. This might be of great help while debugging editor macros to trace the execution of the workspaces. For example the sequence :

```
B1
A
O I AM ENTERING B1
.
.
.
.
.
O I AM LEAVING B1
¢F
B0¢B1
```

will trace the execution of workspace 1 and might help considerably in the debugging.

We have seen that a workspace can be viewed as a procedure which can be invoked by means of the ¢B(x) sequence.However, the expressive power of the procedural language (requests) is rather limited. The requests discussed in this section and the following one are aimed at giving fuller expressive power to the editor language by introducing conditions and jumps.

FUNCTION OF "*":

The Test Contents request (*) is used to test if a line in a given range contains a given regular expression. If such a line is found, the remainder of the request line is executed, otherwise it is discarded and execution continues with the following line.

FORMAT:

ad1,ad2*/regexp/other-requests
(the first character after * is taken to be the regular expression delimiter and can be any character not appearing in the regular expression).

DEFAULT:

/regexp/ is taken to mean .,./regexp/
ad*/regexp/is taken to mean ad,ad*/regexp

VALUE OF ".":

Set to the matched line if a match occurred ; set to the line addressed by ad2 otherwise.

FUNCTION OF "?":

The Test Range request (?) is used to test if the current line (".") belongs to a given range of lines. If so, the remainder of the request line is executed, otherwise it is discarded and execution continues with the following line.

FORMAT:

ad1,ad2? other-requests

DEFAULT:

ad? is taken to mean ad,ad?
? is taken to mean ¬1,$? which means that the remainder of the line is executed except if the current workspace is empty.

VALUE OF ".":

unchanged

FUNCTION:

The goto request (>) can only be executed from a workspace. It is used for skipping a number of request lines or for going a number of request lines backwards.

FORMAT:

> [{±}] n or >Lx (see the following page for this latter format)

if the sign is omitted, + is assumed
n is a decimal number giving the number of lines to skip forwards (+) or backwards (-).

VALUE OF ".":

unchanged.

OPERATION:

In the current workspace, a skip is made n lines forwards or backwards. If this causes a positioning before the first line of the workspace a branch to the first line is assumed ; if this causes a positioning after the last line of the workspace, an exit from the workspace is assumed.

EXAMPLE:

A common editing problem is the following :

"For each line that contains string "string1" perform some kind of action".

Assuming that the detail of the action to be performed is contained in workspace 2, the above problem can be solved by

```
^N
.,$*/string1/>+2
>+4
¢B(2)
$?>+2
.+1N>-4
```

The previously defined method of skipping backwards and forwards in the request stream might be inconvenient when the contents of the request stream (in a workspace) are to be altered. In this case all offsets have to be recalculated each time that a line is inserted or deleted.

Labels allow symbolic reference to a request line.

A label is defined as being the sequence :x, where x stands for any character, in the first position of a request line (i.e. columns 1 and 2). Setting a label does not otherwise alter the execution of the statement.

A labelled statement can be referred to in a goto statement as follows :

```
>Lx
```

where x stands for the character defining the label.

The example in the preceding section could thus equally have been written as follows:

```
¬N
:N .,$*/string/>LE
> LX
: E¢B(2)
$?>LX
.+1N>LN
:X
```

MISCELLANEOUS REQUESTS

We have grouped under this header a number of requests which will not be of great use to a first time user of the TEXT EDITOR.
The reading of this section might therefore be deferred until a specific need arises.

Top of Page Request (T)

FUNCTION:

The top of page request T is used to force a skip to the top of a new page on the TEXT EDITOR output report.

FORMAT:

T

DEFAULT:

NONE

VALUE OF ".":

left unchanged

EXAMPLE:

request

```
                    T↑,$P
```

will cause the printing of the whole current workspace, starting at the top of a new page.

FUNCTION:

The split line request is used to break down lines into two or more lines depending on their contents.

FORMAT :

ad1,ad2%/regexp/

(the first character after % is taken to be the regular expression delimiter and can be any character not appearing in regexp)

DEFAULT:

%/regexp/ is taken to mean .,.%/regexp/
ad%/regexp/ is taken to mean ad,ad%/regexp/

VALUE OF ".":

Set to ad2

ACTION:

All lines matched by regexp in the specified range are treated as follows :

- if the match is before the first character in the line or after the last one, no action is taken

- otherwise, the line is broken down as many times as regexp appears in the line. Each resulting line is delimited by the first character matched by regexp.

Before	Request	After
ABC		ABC (match is the 1st character of line)
CDA	¬,$%/A/	CD
BABABA		A
		B
		AB
		AB
		A (match is last of line)

The Concatenate Request (&)

FUNCTION:

The concatenate request is used to concatenate lines which fulfil a certain criterion.

FORMAT:

adr1,adr2&/regexp/

(the first character after & is taken to be the regular expression delimiter and can be any character not appearing in regexp)

DEFAULT:

&/regexp/ is taken to mean .,.&/regexp/
ad&/regexp/ is taken to mean ad,ad&/regexp/

VALUE OF ".":

set to adr2

ACTION:

All lines matched by the regular expression in the given range are merged (i.e. concatenated at the end) with the immediately preceding line. If the matched line is the first in the workspace no action is taken for that line.

EXAMPLE:

```
Before :      -A
              BB
              -C
              -D
              -E
              F
              G
              -H

Request : ¬,$&/¬-/

After:        -A
              BB-C-D-E
              F
              G-H
```

FUNCTION:

The search backwards request (<)is used to search for a given line backwards. That is, starting from the line before the specified one towards the first line of the workspace, then, if no match occurred, from the last line in the workspace towards the specified line.

FORMAT:

ad</regexp/
(the first character after < is taken to be the regular expression delimiter and can be any character not appearing in the regular expression)

DEFAULT:

</regexp/ is taken to mean .</regexp/

VALUE OF ".":

set to the matched line if found, to the line addressed by ad otherwise.

EXAMPLE:

```
- Before -
              A
              B
              C
              D
"." ------->  D
              E

- Request -
              </B/

- After -
              A
"." ------->  B
              C
              D
              D
              E
```

The Special Control Request (Y)

FUNCTION:

The user of the editor might wish to control the contents of the report listing produced by the TEXT EDITOR or its behaviour when an error occurs. This is achieved by means of the Y Request.

FORMAT:

YM	YE	{1 3}
YV		
YB		
YL	YR	{1 3}
YF		
YN		
YS	YP	{1 3}
YW		

MEANING:

- YV (for Verbose) will produce a printing of all target lines in Substitute requests before the substitution is made, thus giving a trace of the modified lines. YM (for Mute) is the default setting which produces no trace.

- In normal mode, request and input lines are echoed on the report listing. This trace may be suppressed by means of the YB (for Brief) request. Default setting is YL (for Long) which produces the echo.

- YN (for Trace oN) will produce a trace of request lines executed from a workspace, thus providing a useful tool for debugging editor macros. Default setting YF (for Trace oFf) does not produce this trace.

- YS (for Strong) will produce a severity 3 diagnostic for all errors. YW (for Weak) will produce a severity 1 diagnostic for some user errors. Irrecoverable errors or system failures will still be reported with severity 3. Default setting is YS (severity 3 for all errors).

- YE1 simulates the occurrence of a severity 1 error ; YE3 simulates a severity 3 error.

- YR3 resets to zero the severity 1 and 3 diagnostic counters;
  YR1 resets to zero the severity 1 diagnostic counter.

- YP3 is used to specify that any subsequent W or Z request is to be rejected if a severity 3 diagnostic has previously occurred. YP1 is used to specify that any subsequent W or Z request is to be rejected if a severity 3 or 1 diagnostic has previously occurred. Default option is never to reject W or Z requests even after a severity 1 or 3 diagnostic.

ESCAPE SEQUENCES

The Escape sequence mechanism is a device provided to alter the way the editor interprets its input stream. Two of those sequences were already discussed earlier in this section:

¢F, to indicate the end of input mode
¢B, to denote workspace invocation

The other escape sequences will be discussed in this section. They are all introduced by means of symbol ¢ followed by a distinctive letter which might be followed by one or more complementary characters.

Protection (¢C)

The protection escape sequence is used for entering text that might otherwise be treated as an escape sequence. For example, to enter sequence ¢B(3) as a literal text,¢¢CB(3) should be entered in order that the sequence is not considered as a workspace invocation.

The general rule is that protection sequences ¢C are eliminated in all contexts after other escape sequences have been treated. Protection sequences may be nested at any depth to provide successive protection against escape sequence processing :

$$\underbrace{¢¢.....¢}_{n} \qquad \underbrace{CC.....C}_{n}$$

Hexadecimal Escape (¢X)

A user may wish to work with characters for which no graphics exist. This can be achieved by means of the ¢X escape sequence.

In all contexts, sequence ¢Xhh(where hh stands for 2 hexadecimal digits) is treated as if the single character whose internal hexadecimal representation is hh had been entered. Note that two hexadecimal digits must be present (i.e.¢X0F and not¢XF). Any sequence where syntax is incorrect (i.e. :¢XF, ¢X0Z,etc..) is treated as a literal string input.

The meaning of the ¢X sequence can be overridden by means of the
¢C protection escape sequence(i.e.:¢¢CX)

The attached table summarizes the syntax and use of the TEXT EDITOR functions.

TABLE 12-1. SUMMARY OF EDIT FUNCTIONS

Request	Meaning	Syntax	Default	Values of "."
Space	Locate	ad adA	none	set to ad
A	append	text F	.A	last appended
B	change work space	B(x) ad1,ad2 C	none	unchanged
C	change	text F	.,.C	last changed
D	delete	ad1,ad2 D	.,.D	after ad2
E	end file output	E	none	unchanged
F	file output	F(x)	none	unchanged
G	apply x to all lines with /re/	ad1,ad2Gx/re/	,$Gx/re/	set to ad2

TABLE 12-1 (CONT). SUMMARY OF EDIT FUNCTIONS

Request	Meaning	Syntax	Default	Values of "."
I	insert	adI text F	.I	last inserted
K	copy	ad1,ad2K(x)	.,.K (x)	set to ad2
L	print with line number	ad1,ad2L	.,.L	set to ad2
M	move	ad1,ad2M(x)	.,.M (x)	set to ad2
N	no operation	adN	.N	set to ad
()	output message	() text	none	unchanged
P	print	ad1,ad2P	.,. P	set to ad2
Q	quit	Q	none	lost
R	read	adRname	$Rname	last line read
S	substitute	ad1,ad2S/re/st/	.,.S/re/st/	set to ad2
T	top of page	T	none	unchanged
V	apply x to all lines except those with /re/	ad1,ad2Vx/re/	¬,$Vx/re/	set to ad2
W	write	ad1,ad2Wname	$Wname	unchanged

TABLE 12-1 (CONT). SUMMARY OF EDIT FUNCTIONS

Request	Meaning	Syntax	Default	Values of "."
X	print workspaces status	X	none	unchanged
Y	special	YB YS YF YW YL YP {1 3} YE {1 3} YM YR {1 3} YN	none	unchanged
Z	overwrite	ad1,ad2Zname	,$zname	unchanged
=	print line number	ad =	. =	set to ad
"	comment	ad " text	. " text	set to ad
<	search backwards	ad</re/	.</re/	set to matched line or ad
*	test contents	ad1,ad2*/re/	.,$*/re/	set to matched line or ad2
>	goto	>[±] >Lx	none	unchanged

TABLE 12-1 (CONT). SUMMARY OF EDIT FUNCTIONS

Request	Meaning	Syntax	Default	Value of "."
:	define label	: x	none	unchanged
?	test range	ad1,ad2?	⌝,$?	unchanged
#	count lines	ad1,ad2# /re/	.,.#/re/	set to ad2
%	split lines	ad1,ad2%/re/	.,.%/re/	set to ad2
&	concatenate lines	ad1,ad2&/re/	.,. &/re/	set to ad2
¢	special escape	¢B(x) ¢C ¢F ¢Xhh	none	depending on B (x) unchanged

JK

L

M

N

O

P

Q

R

S

U

V

W

X

Y

Z

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 64)
LIBRARY MAINTENANCE REFERENCE MANUAL
ADDENDUM B

ORDER NO. AQ28-01B

DATED MARCH 1980

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

21600, 1.5978, Printed in U.S.A.

AQ28, Rev