THIS MANUAL IS CURRENT FOR MDBS 3.06 SCREEN MASTER 1.02

MDBS
MLINK Reference Manual
(Version 1.06)

Micro Data Base Systems, Inc.

P.O. Box 248

Lafayette, Indiana 47902

USA

December 1983

NEWS RELEASES, VERSIONS, AND A WARNING

Any programming endeavor of the magnitude of the MDBS software will necessarily continue to evolve over time. Realizing this, Micro Data Base Systems, Inc., vows to provide its users with updates to **this version** for a nominal handling fee. New versions of MDBS software will be considered as separate products.

Our products have been produced with a very substantial investment of capital and labor, to say nothing of the years of prior involvement in the data base management area by our principals. Accordingly, we are seriously concerned about any unauthorized copying of our products and will take any and all available legal action against illegal copying or distribution of our products.

I. MLINK DESCRIPTION

A. Overview

MLINK is a utility program that is executable under the following operating systems: CP/M, MP/M, CP/M86, CP/M86, PCDOS, MSDOS. When MLINK is invoked, it links together relocatable object modules. MLINK accepts REL, ERL, and IRL object files. The result is an executable file. Under CP/M or MP/M, this is a .COM file. It is a .CMD file for CP/M86 or MP/M86. Under PCDOS or MSDOS, a .EXE file is generated.

B. Using MLINK: Batch Mode

Depending on the operating system environment, MLINK is provided on either one or three files:

MLINK.COM, MLINK1.OVL, MLINK2.OVL under CP/M and MP/M
MLINK.CMD under CP/M86 and MP/M86
MLINK.EXE under PCDOS and MSDOS

The two overlay files (if present) must be on the default drive or drive A.* To execute MLINK, a command line of the following form is entered:

MLINK filel [file2 file3 ... filek]

where filel, file2, ... and filek are names of REL, ERL, and/or IRL files. These must be fully qualified file names. If an extension is not specified for a file, then the .REL extension is assumed.

MLINK generates two files on the same drive as filel: filel.COM (or filel.CMD or filel.EXE) and filel.SYM. The former contains executable object code; the latter contains a symbol table. If an alternative name or drive is desired for the COM (or CMD or EXE) and SYM files, the -O option can be used:

MLINK -Oaltfile filel [file2 file3 ... filek]

where altfile is the alternative file name. If no extension is specified for altfile, the .COM (.CMD or .EXE) extension is used for the linked output file. The .SYM extension is used for the symbol table.

The ordering of files in the command line is important if one or more of the files is a library of object modules. The files are processed in the order given on the command line. Since MLINK satisfies forward references, an external reference must be unsatisfied when MLINK searches the library containing the object module for that external. In other words, the library file, containing an external that is referenced in another file, should follow that other file in the command line.

MLINK performs selective linking with a library file when the library file name is prefaced with -L. Only those object modules in the library file that are required to satisfy undefined externals will be linked. For instance,

MLINK filel -Lfile2

generates the linked filel.COM (filel.CMD or filel.EXE), which does not include any object modules from file2 that are not referenced in filel.

---

*In looking for an overlay file, the current user area of the default drive is searched first. If it is not found then drive A is searched (if user areas are supported, then only user area 0 of drive A is searched).

Several other optional arguments can be included on the command line. As noted in the following descriptions, some of these arguments are pertinent only for certain operating systems or programming languages.

-Ahhhh where **hhhh** is a hex number that specifies the address where the linker's output file is to be loaded. When this optional argument is omitted a load address of 100 hex is used. This argument can be used only under CP/M or MP/M.

-A<hhhh where hhhh is a hex number that specifies the address below which MLINK's output file is to be loaded. After MLINK determines the size of the object program to be generated, MLINK assigns a starting address (on a 256 byte boundary) such that the program will reside below hhhh. This argument can be used only under CP/M or MP/M.

-B This option is used when linking an overlay. It tells the overlay loader that the overlay is not to be re-loaded at run time if it is already in memory.

-C This option causes a "compact" output file to be produced by first assigning all initialized program, common, and data segments. It then assigns all uninitialized program, common and data segments. Uninitialized segments are not written to the output file. This option can be used only under CP/M or MP/M and it may cause improper operation of code produced by certain compilers.

-E This option must be specified when linking C' programs. It inserts ?CRT0 (C' runtime startup module) into the symbol table as an unsatisfied external. This option must be specified before LIBC is searched. -E does not have to be specified when linking C' overlays.

-Esymbol where **symbol** consists of from one to six characters. This option has the effect of inserting the indicated symbol name into the symbol table as an undefined external.

-Esymbol=hhhh where **symbol** consists of from one to six characters and **hhhh** is a hex address. This option defines the indicated **symbol** to be an entry point with address **hhhh.**

-Iincfile where **incfile** is the name of a file containing one or more MLINK arguments. All arguments from **incfile** are included in the command line in place of the -I argument. The contents of **incfile** are free form and can extend over many lines. MLINK arguments in **incfile** are separated by blanks. Any line in **incfile** that begins with a semi-colon (;) is ignored by MLINK. The -I argument can be nested within any **incfile.**

-R This option must be specified when linking the "root" file of an overlayed program. Note that this argument must occur before the search of the library containing the overlay loader (?OVLD). For any language other than C', this means that -R must precede -LOSCPM (under CP/M, MP/M), -LOSCPM86 (under CP/M86, MP/M86), or -LOSMSDOS (under PCDOS, MSDOS).

-Rffff This option must be specified when linking an overlay. ffff indicates the name of the "root" file and its symbol table file. This argument must occur before any -L arguments.

-Sstring where **string** is a sequence of characters. This option suppresses the writing of any symbol (to the symbol table file) that begins with a character in the indicated **string.** For instance, -S$? suppresses the writing of any symbols beginning with ? or $. If no **string** is specified, a new symbol table file will not be created.

-Uhhhh where **hhhh** is the address given to all unsatisfied externals at the end of pass 1. If **hhhh** is omitted, each unsatisfied external is given a unique address outside any areas assigned by the linker.

-V This is a toggle that causes MLINK to alternate between verbose mode and terse mode. Verbose mode causes MLINK to display the nature of its processing as it executes. The first use of a -V option turns on the verbose mode, the second -V turns it off, the third -V turns it on, etc.

-X This option suppresses the automatic generation of a jump to the transfer address that may occur at the beginning of the output file.

-Xhhhh where hhhh is a hexadecimal number. This option defines the transfer address (the address at which program execution begins: ?XFER?) to be hhhh. If this option appears before modules with the transfer address, then hhhh overrides the transfer address specified in these modules and MLINK will issue a multiple ?XFER? definition diagnostic.

-Ystfile where **stfile** is a ZSID type symbol table file. This option enters the symbols from **stfile** into MLINK's symbol table.

-Z This option initializes (to 0) all uninitialized areas on the output file generated by MLINK.

Unless otherwise noted, the optional arguments can be used in any order. For instance:

```
MLINK -V -A0200 -OC:PRG PRG -LMDBSL -U
```

C. Using MLINK: Interactive Mode

To execute MLINK in interactive mode, a command line of the following form is entered:

```
MLINK
```

The following response is displayed:

```
Interactive mode: enter MLINK arguments.
To start pass 2, enter a null line.
>
```

The > symbol is the interactive mode prompt. Any argument that can be used in regular mode can also be used in interactive mode. However, since interactive mode is normally verbose, the first -V will turn off the display of verbose messages, the second -V causes verbose messages to be displayed again, etc. To exit the interactive mode of MLINK, press the carriage return in response to an interactive mode prompt.

D. MLINK Operation

MLINK is a two-pass linker that processes the command line in a left to right fashion. The first pass constructs the symbol table, containing entries for each of three types of segments: program segments, common segments, and data segments. MLINK determines the size of each segment and assigns an absolute address to each. When determining the size of a segment, the largest definition of that segment is used (multiple definitions of common segments are legal). Program segment addresses are assigned first, common segments are then assigned (blank common, if it is used, is the first of these common segments), and data segment addresses are assigned last. MLINK then passes through the symbol table to resolve all entry points, by transforming offsets into absolute addresses. The second pass examines needed files and object modules to generate executable object code.

Appendix B shows the minimal MLINK command line arguments for programs in various languages.

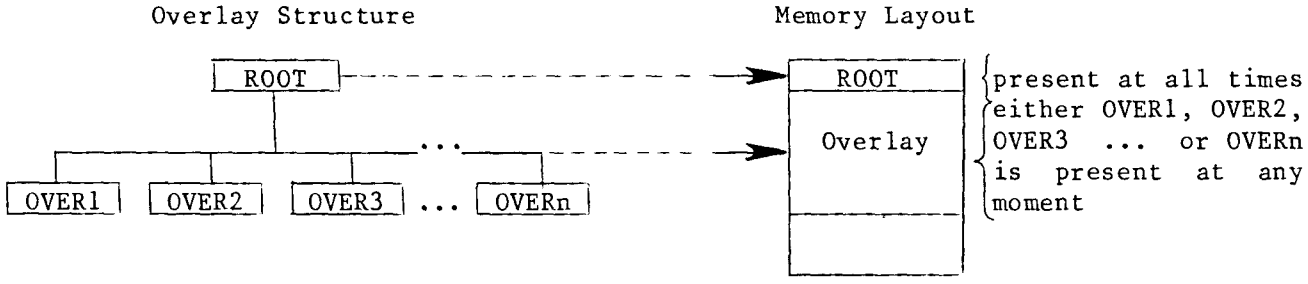
At any time during MLINK execution (batch or interactive), MLINK can be aborted by typing Control-C or Escape.

E. Overlays

The initial executable file generated by MLINK is called the root file; it has a .COM (.CMD or .EXE) extension. The overlay files generated by MLINK have .OVL extensions.

During execution of your linked program an invoked overlay file must be on the default drive or drive A.* The overlay loader will load the specified file (e.g., OVER) and initiate execution of the first routine in that file. Therefore, for simplicity, it is advisable to give this routine the same name that the overlay file has (e.g., OVER).

1. Single-Level Overlay Structures

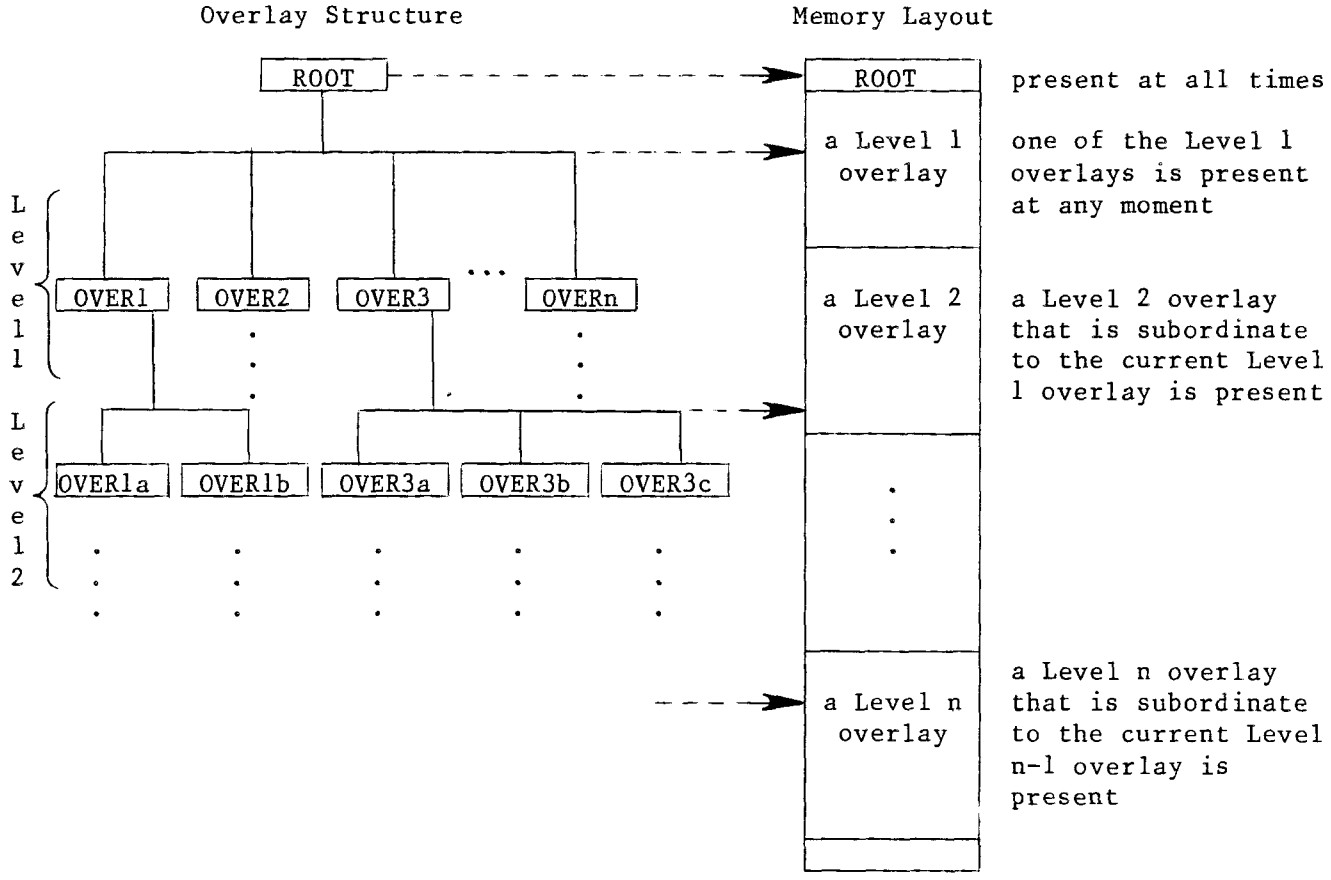


During execution, the root is always resident. Overlays are loaded as required, with each overlay overwriting the previously resident overlay. Thus overlays are convenient in memory constrained environments, where the total program length exceeds the available memory. Such a program can be segmented into a root and a group of overlays, no two of which need to be resident simultaneously. When MLINK detects a call in the root for an overlay (e.g., OVER), it generates a call to the overlay loader ?OVLD. Examples of how to use MLINK for handling overlays in various languages are shown in Appendix C.

---

* In looking for an overlay file, the current user area of the default drive is searched first. If it is not found then drive A is searched (if user areas are supported, then only user area 0 of drive A is searched).

2. Multi-Level Overlay Structures



MLINK imposes no maximum on the number of levels in an overlay structure. During execution, the root is always resident. At any given moment, one overlay from each level in the overlay structure can also be resident. The overlay that is resident for a particular level must be a subordinate of the next higher level's resident overlay. For example, OVER1 must be resident before OVER1a or OVER1b can be invoked during execution. Dynamic memory allocation occurs when overlay files generated by MLINK are loaded, thereby avoiding excessive memory requirements. Thus, OVER3 may be very large compared to OVER1 while its subordinate overlays are small relative to those of OVER1. Appendix C shows examples of MLINK usage for handling multi-level overlays in various languages.

3. Overlay Load Errors

As ?OVLD attempts to load an overlay, it can detect any of three possible errors. The diagnostics issued by ?OVLD and an explanation of each appear below.

Overlay error on file.OVL **: incompatible overlay**
?OVLD attempted to load the indicated file.OVL. However, the copy of file.OVL presently on the default drive is incompatible with the version of the root (or a higher level overlay) that is being executed. If file.OVL does not exist on the default drive, then the file.OVL on drive A is inconsistent with the executing root.

Overlay error on file.OVL **: file not accessible**
The indicated file.OVL is not presently on either the default drive or drive A.

Overlay error on file.OVL **: read error**
When ?OVLD attempted to load file.OVL an error was detected while loading a header, data, or code area of file.OVL. This may be caused by a disk read error. It may also be due to a corrupt copy of the file.OVL (e.g., due to bad media).

For non-C' programs, the program exits to the operating system after the diagnostic is displayed. For a C' program the diagnostic is displayed. Then, a signal SIG_OVL is generated which can be trapped by the C' program (see the signal function in the C' library), but which cannot be ignored. If the program attempts to ignore it, control returns to the operating system.

F. Miscellaneous Notes for Advanced MLINK Usage

1. If split I&D (instruction and data areas) is permitted by the operating system, then MLINK generates executable split I&D files.

2. The value of ?NEXT? is the first free memory data address (in hex). If split I&D is permitted then the value of ?NXTC? is the first free memory code address (in hex).

3. The value of ?XFER? is the address (in hex) where execution of the linked program will begin. This is either a transfer address specified in one of the .REL files or the lowest code-relative reference point. If there is no transfer address and no code-relative entry points are defined, then ?XFER? is undefined.

4. If .END. is an unsatisfied external reference at the end of pass 1, then it is set to ?NEXT?.

5. A program being linked can define three entry points (?MEMRY, $MEMRY, .NEXT.) which are treated specially by MLINK. At the end of pass 2, MLINK puts the value of ?NEXT? into the word existing at any of the three special entry points.

6. For IRL files, MLINK skips the IRL header.

II. MLINK MESSAGES

A. Normal Operation Messages

The following messages can appear in the course of normal MLINK operations:

MDBS MLINK V1.xx (xxx)
(C) COPYRIGHT 1983, Micro Data Base Systems, Inc.
Lafayette, IN 47902

Appearance of this message identifies the version of MLINK and indicates that MLINK has successfully begun to execute.

nn overlays called

The -R option was specified, and the entry points listed were not defined in this link. Calling to one of these entry points will execute a linker generated implicit call to the overlay loader.

Xfer = hhhh

Execution of the linked program will begin at address hhhh (hex). This message appears only when a transfer address has been specified in a relocatable file.

Org = hhhh

The first address of the output file is hhhh (hex). This message occurs when linking overlays or when the -A<hhhh argument is used. If using the BASCOM chaining features, this is the address at which blank common begins (i.e., the address specified in the BCLOAD file).

Next = hhhh

The first free memory address (following those assigned to program, common and data segments) is hhhh (hex). This message indicates the end of MLINK's first pass.

ffff linking complete.

This message indicates that the linker has finished without encountering errors which would abort the link. Some non-fatal errors may have been encountered. ffff is the name of the output file created by MLINK.

B. Verbose Mode Messages

When the -V option is used to toggle verbose mode, the following kinds of verbose mode messages may be displayed in addition to normal messages described above.

Code = hhhh

The start of the area that holds all modules' program segments is hhhh (hex).

Common = hhhh

The start of the area that holds all modules' common segments is hhhh (hex).

Data = hhhh

The start of the area that holds all modules' data segments is hhhh (hex).

ffff:

The name of a file (ffff) and all modules selected from ffff are displayed. For each file this display occurs once during the first pass scan and once during the second pass link.

Initialized
Uninitialized

These messages precede the Code, Common, and Data messages for their respective areas. See the -C argument in Section I-B of this manual.

nn Modules:

nn (decimal) modules have been selected for linking from the specified file.

nnK table space used:

nn (decimal) K bytes has been dynamically allocated by MLINK for buffers and tables.

C. MLINK Error Messages

If an error occurred in a particular file (rather than in the command line), then the error message is prefaced by the name of that file:

ffff:

If such an error occurred in a REL, ERL or IRL file, then the affected module's name (if known) is displayed in square brackets after the file name and before the message itself.

ffff: [module: mmmm]

The possible error messages are shown below. Note that some errors are fatal, while others are non-fatal. Processing is aborted and the user is returned to command mode in the event of a fatal error. Processing continues for non-fatal errors. When processing halts, the message

nn Errors

appears, where nn is the number of error messages MLINK issues during the link.

-A: bad address non-fatal

The A argument did not contain a hexadecimal number or the number contained a bad digit. The argument is ignored and linking continues.

Address outside seg-desc segment non-fatal

A byte outside of the segment currently being worked on was initialized or there is a reference to an address outside of the segment. The segment is indicated by seg-desc which is either: code, data, or a common block name enclosed between / and /. Generally, this condition is caused by incorrect assembler or compiler operation (or by a corrupt REL file) and it typically produces unpredictable results.

Bad argument -z non-fatal

-z is not a valid MLINK argument and is ignored. Processing continues. See section I-B of this manual for a description of valid MLINK arguments.

Bad header non-fatal

The header on an IRL file was not of proper format. This usually indicates a corrupt IRL file. On CP/M, this may have been caused by copying the file with PIP without using the O option of PIP. This file is ignored and linking continues.

Cannot load address xxxx non-fatal

The specified address xxxx is lower than the origin of the output file. This load is ignored and linking continues.

Can't create output file fatal

Under CP/M, this is generally caused by a full disk directory. In rare instances, this is caused by insufficient memory.

Can't create symbol table file fatal

Under CP/M, this is generally caused by a full disk directory. In rare instances, this is caused by insufficient memory.

Can't load ffff fatal

Overlay file ffff was not present (see section I-B of this manual) or could not be read.

Can't open ffff non-fatal (pass 1) / fatal (pass 2)

The specified file ffff could not be read. This results from a non-existent file, an error in the file name, or in rare instances, insufficient memory (this occurs when input buffer space cannot be allocated). This file is skipped and linking continues (during pass 1).

Can't open root file fatal

When attempting to link an overlay, the root file specified with -Rffff could not be opened for read/write access.

Chain too long fatal

An external or address chain exceeded 1000 items. This indicates a corrupt REL file or more than 1000 references to a given external within a particular module.

Code size redefined non-fatal

More than one code size declaration was encountered in the specified module, generally a corrupt REL file. The largest definition is used and linking continues.

Common block /bbbb/ size mismatch non-fatal

MLINK encountered a definition of the specified common block, bbbb, smaller than the largest definition specified for that common block. MLINK uses the largest.

Data size redefined non-fatal

More than one data size declaration was encountered in the specified module, generally a corrupt REL file. The largest definition is used and linking continues.

-E: name required non-fatal

A symbol name did not immediately follow the E argument. The argument is ignored and linking continues.

End of file unexpected non-fatal (pass 1)
fatal (pass 2)

The end-of-file marker occurred within module mmmm on file ffff. This is usually an indication that this file has been corrupted. On CP/M, this may have been caused by copying the file with PIP without using the O option of PIP. This module is ignored and linking continues (during pass 1).

***** FATAL ERROR ***** fatal

MLINK is aborting upon encountering a fatal error.

Insufficient memory fatal

MLINK requires more memory to construct all tables required by this link.

***** INTERRUPTED ***** fatal

MLINK is aborting because the Control-C key or Escape key was pressed.

Multiple definition - ssss non-fatal

Entry point ssss is multiply defined. The first definition is used and linking continues.

Multiple definition - ?XFER? non-fatal

More than one transfer address definition was encountered. The first definition is used and linking continues.

No common block selected - ssss non-fatal

An attempt was made to define an entry point ssss into a common block without first selecting the common block. This generally indicates a corrupt REL file. "Absolute" mode is used for this reference and linking continues.

No modules linked fatal

The link directives did not specify linking of any modules which could be properly accessed by the linker. This is usually due to the fact that one or more of the files specified could not be opened (in this case this message would be preceeded by error messages specifying those files which could not be opened). The linker aborts without creating a new output or symbol file.

No output file-name fatal

No output file name was specified by the directives. If this occurs, the O option must be used to specify a name.

Overlay error on MLINK1.OVL or MLINK2.OVL : **incompatible overlay** fatal

?OVLD attempted to load the indicated MLINK1.OVL or MLINK2.OVL. However, the copy of MLINK1.OVL or MLINK2.OVL presently on the default drive is incompatible with the version of the root that is being executed. If MLINK1.OVL or MLINK2.OVL does not exist on the default drive, then the MLINK1.OVL or MLINK2.OVL on drive A is inconsistent with the executing root.

Overlay error on MLINK1.OVL or MLINK2.OVL : **file not accessible** fatal

The indicated MLINK1.OVL or MLINK2.OVL is not presently on either the default drive or drive A.

Overlay error on MLINK1.OVL or MLINK2.OVL : **read error** fatal

When ?OVLD attempted to load MLINK1.OVL or MLINK2.OVL an error was detected while loading a header, data, or code area of file.OVL. This may be caused by a disk read error. It may also be due to a corrupt copy of the MLINK1.OVL or MLINK2.OVL (e.g., due to bad media).

Read error non-fatal (pass-1)
fatal (pass-2)

This indicates that the specified file is corrupt. This file is skipped and linking continues (during pass 1).

Read error on output file fatal

An error occurred when attempting to "page-in" a part of the output file. This should not occur.

Syntax error non-fatal

The specified text file contained improper syntax. For instance, when linking a BASCOM program the syntax of the BCLOAD file was incorrect. Consult the language manual (e.g., BASCOM) for correct syntax.

nn Unsatisfied externals non-fatal

No entry point definitions were found for the specified externals (a list of names follows this message). This is usually caused by missing or misnamed subroutines, or by out-of-order libraries. If the -U argument was not used, MLINK automatically enters the interactive mode, displaying the message

```
Type ^C to abort, or CR to start pass 2,
or enter further MLINK arguments now:
>
```

Typing Control-C aborts processing. Pressing the carriage return causes all unsatisfied externals to be given an address of zero and linking continues. Therefore, during execution, calling such a routine will result in a call to address zero (under CP/M or MP/M this causes the program to exit).

Version mismatch fatal

One or more of the MLINK overlay files is not of the same version as the MLINK root. Make fresh copies of the MLINK root and overlay files from the distribution disk to ensure a version match.

Write error on output file fatal

Under CP/M, this is generally caused by a full disk.

Write error on symbol table file fatal

Under CP/M, this is generally caused by a full disk.

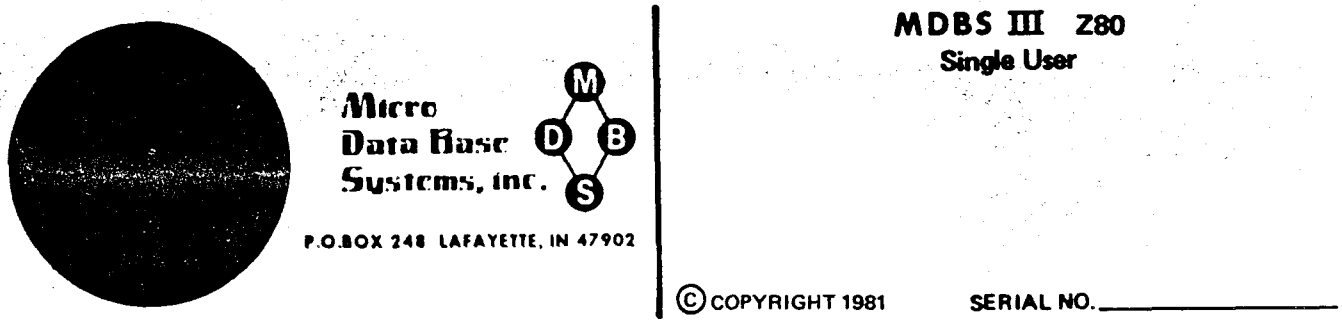
This page intentionally left blank.

Do not purchase or otherwise accept MDBS III software products not bearing the official colored MDBS diskette/tape label. The serial number on the label uniquely identifies the system licensee. If you are approached by a person or organization attempting to illegally distribute MDBS products, please contact

Micro Data Base Systems, Inc.
P.O. Box 248
Lafayette, IN 47902
(317) 448-1616 Telex 209147 ISE UR

We appreciate your assistance in preventing software piracy.

Sample of MDBS III diskette/tape label:



MDBS MLINK MANUAL

Appendix A

Relocatable Object File Format

Appendix A

Microsoft REL is a format for RELocatable object files, usually used for 8080/Z80 object code under CP/M. It is a bit-stream, which means that the linker directives (here called items) occur on bit boundaries, rather than byte or word boundaries (except for a couple of special items mentioned later).

Relocatable_Object_File_Format:

Each item in the bit stream consists of a header bit(s), and depending on the item type, other fields (expressed here in BNF grammar):

```
byte := bit bit bit bit bit bit bit bit /* 8 bits */
word := byte byte /* low byte first */
/* address field */
a-field := 0 0 word |          /* absolute */
           0 1 word |       /* code-relative */
           1 0 word |       /* data-relative */
           1 1 word         /* common-relative */
/* name field */
b-field := 0 0 0 |          /* header bits are name char count */
           0 0 1 byte |
           0 1 0 byte byte |
           0 1 1 byte byte byte |
           1 0 0 byte byte byte byte |
           1 0 1 byte byte byte byte byte |
           1 1 0 byte byte byte byte byte byte |
           1 1 1 byte byte byte byte byte byte byte
filler :=  (null) | bit          /* enough bits to force next item */
           bit bit |          /* to byte boundary */
           bit bit bit |
           bit bit bit bit |
           bit bit bit bit bit |
           bit bit bit bit bit bit |
           bit bit bit bit bit bit bit
```

Given all this, the definitions are:

```
REL file            := L_ENDF | module-group L_ENDF

module-group              := module | module module-group

module                    := L_EPRG | item-group L_EPRG

item-group                := relitem | relitem item-group

relitem                   := L_ESYM | L_SCOM | L_PNAM |
                        L_LSCH | L_DCOM | L_CEXT |
                        L_DENT | L_MOFF | L_POFF |
                        L_DSIZ | L_SLOC | L_CADR |
                        L_PSIZ | L_EPRG | L_ENDF |
                        L_WORD | L_BYTE
```


L_ESYM "entry symbol" := 1 0 0 0 0 0 0 b-field

(b-field is name of entry point contained in this module. This item is used during library searches to determine whether this module satisfies an unsatisfied external.)

L_SCOM "select common block" := 1 0 0 0 0 0 1 b-field

(b-field is the name of a common block to use for common-relative references after this item.)

L_PNAM "module name" := 1 0 0 0 0 1 0 b-field

(b-field is the name used by LIB to identify the module. Note: P stands for program, but this really means module.)

L_LSCH "library search file" := 1 0 0 0 0 1 1 b-field

(b-field is the name of a REL file to be searched after all the linker directives are processed if there are still unsatisfied externals.)

L_DCOM "declare common block" := 1 0 0 0 1 0 1 a-field b-field

(b-field is the name of a common block whose size is specified by the a-field (the mode is ignored).)

L_CEXT "chain external" := 1 0 0 0 1 1 0 a-field b-field

(a-field is the head address of the chain, and the b-field is the name of the entry point whose address is to replace into each member of the chain (chain is terminated by an absolute zero entry.)

L_DENT "declare entry point" := 1 0 0 0 1 1 1 a-field b-field

(b-field is the name of an entry point, and a-field is its address.)

L_MOFF "External - offset" := 1 0 0 1 0 0 0 a-field

(a-field is offset to be subtracted from value at this address, after all chaining has been done.)

L_POFF "External + offset" := 1 0 0 1 0 0 1 a-field

(a-field is just like in L_MOFF, but added instead of subtracted.)


```
L_DSIZ "Data area size"        := 1 0 0 1 0 1 0 a-field
```

(a-field is data area size, mode is ignored.)

```
L_SLOC "Set loading location" := 1 0 0 1 0 1 1 a-field
```

(a-field specified the location of the next byte/word is to loaded.)

```
L_CADR "Chain address"         := 1 0 0 1 1 0 0 a-field
```

(a-field is starting address of chain, and present loading location is to replace into each member of the chain (chain is terminated by an absolute zero entry).)

```
L_PSIZ "Module size"           := 1 0 0 1 1 0 1 a-field
```

(a-field is module code area size, mode is ignored.)

```
L_EPRG "Module end, with transfer address"
                               := 1 0 0 1 1 1 0 a-field filler
```

(a-field is the transfer address; absolute zero means no transfer address.)

```
L_ENDF "End of file"           := 1 0 0 1 1 1 1

L_BYTE "Load a byte"           := 0 byte

L_WORD "Load a word"           := 1 0 1 word /* code-relative */
```

In a RELITEM, "ri_type" is the item type, "ri_val" is the a-field (address field) if the item has one, "ri_mode" is the address mode of the a-field, and "ri_name" is a null-terminated array containing the b-field (name field) if the item has one. For the cases of L_BYTE and L_WORD, "ri_val" is the byte/word value, with "ri_mode" the addressing mode for L_WORD.

MDBS MLINK MANUAL

Appendix B

Minimal MLINK Command Line Arguments

Appendix B

PROG is the name of a file holding the program to be linked.

Assembly Language

MLINK PROG

C' Compiler

MLINK -CE PROG

BASCOM-80

MLINK PROG -LBASLIB

CB-80

MLINK POG -LCB80.IRL

COBOL-80

MLINK PROG -LCOBLIB -LTDRIVER

FORTRAN-80

MLINK PROG -LFORLIB

PASCAL MT+ Compiler

MLINK -ZX PROG.ERL -LPASLIB.ERL

PLI-80

MLINK PROG -LPLI80.IRL

MDBS MLINK MANUAL

Appendix C

Using MLINK with Overlaying

Appendix C

Assembly Language

Single-level overlays:

```
MLINK -R ROOT LOSCPM
MLINK -Z -RROOT OVER1
MLINK -Z -RROOT OVER2
```

Multi-level overlays:

```
MLINK -R ROOT -LOSCPM
MLINK -RROOT OVER1
MLINK -RROOT OVER2
MLINK -RROOT -YOVER2 OVER2a
MLINK -RROOT -YOVER2 OVER2b
MLINK -RROOT -YOVER2 -YOVER2b OVER2bx
MLINK -RROOT -YOVER2 -YOVER2b OVER2by
```

C' Compiler

Single-level overlays:

```
MLINK -CE -R ROOT
MLINK -RROOT OVER1
MLINK -RROOT OVER2
```

Multi-level overlays:

```
MLINK -CE -R ROOT
MLINK -RROOT OVER1
MLINK -RROOT OVER2
MLINK -RROOT -YOVER2 OVER2a
MLINK -RROOT -YOVER2 OVER2b
MLINK -RROOT -YOVER2 -YOVER2b OVER2bx
MLINK -RROOT -YOVER2 -YOVER2b OVER2by
```

FORTRAN-80

Single-level overlays:

```
MLINK -R ROOT.ERL -LFORLIB -LOSCPM
MLINK -RROOT OVER1.ERL -LFORLIB
MLINK -RROOT OVER2.ERL -LFORLIB
```

PASCAL MT+ Compiler

Single-level overlays:

```
MLINK -ZX -R ROOT.ERL PINI.ERL -LPASLIB.ERL -LOSCPM
MLINK -Z -RROOT OVER1.ERL -LPASLIB.ERL
MLINK -Z -RROOT OVER2.ERL -LPASLIB.ERL
```

Multi-level overlays:

```
MLINK -ZX -R ROOT.ERL PINI.ERL -LPASLIB.ERL -LOSCPM
MLINK -Z -RROOT OVER1.ERL -LPASLIB.ERL
MLINK -Z -RROOT OVER2.ERL -LPASLIB.ERL
MLINK -Z -RROOT -YOVER2 OVER2a.ERL -LPASLIB.ERL
MLINK -Z -RROOT -YOVER2 OVER2b.ERL -LPASLIB.ERL
MLINK -Z -RROOT -YOVER2 -YOVER2b OVER2bx.ERL -LPASLIBERL
MLINK -Z -RROOT -YOVER2 -YOVER2b OVER2by.ERL -LPASLIB.ERL
```


DOCUMENTATION COMMENT FORM

MDBS Document Title:_____

We welcome and appreciate all comments and suggestions that can help us to improve our manuals and products. Use this form to express your views concerning this manual.

Please do not use this form to report system problems or to request materials, etc. System problems should be reported to **MDBS** by phone or telex, or in a separate letter addressed to the attention of the technical support division. Requests for published materials should be addressed to the attention of the marketing division.

Sender:

_____ _____
(name) (position)

_____ _____
(company) (telephone)

_____
(address)

_____
(city, state, zip)

COMMENTS:

Areas of comment are general presentation, format, organization, completeness, clarity, accuracy, etc. If a comment applies to a specific page or pages, please cite the page number(s).

_____

_____

_____

_____

_____

_____

_____

_____

Continue on additional pages, as needed. Thank you for your response.

MDBS

MLINK REFERENCE MANUAL

(VERSION 1.XX)

FOR CP/M AND MP/M

Micro Data Base Systems, Inc.

P. O. Box 248

Lafayette, Indiana 47902

USA

February 1982
(Revised August 1982 for Version 1.03)
(Revised November 1982 for Version 1.03h)

NEWS RELEASES, VERSIONS, AND A WARNING

Any programming endeavor of the magnitude of the MDBS software will necessarily continue to evolve over time. Realizing this, Micro Data Base Systems, Inc., vows to provide its users with updates to **this version** for a nominal handling fee. New versions of MDBS software will be considered as separate products.

Our products have been produced with a very substantial investment of capital and labor, to say nothing of the years of prior involvement in the data base management area by our principals. Accordingly, we are seriously concerned about any unauthorized copying of our products and will take any and all available legal action against illegal copying or distribution of our products.

I. MLINK DESCRIPTION

A. Overview

MLINK is a utility program that is executable under the CP/M and MP/M operating systems. When MLINK is invoked, it links together relocatable object modules. The result is a .COM file that can be executed under CP/M or MP/M. MLINK accepts REL, ERL, and IRL object files.

B. Using MLINK: Batch Mode

MLINK is provided on the files named MLINK.COM, MLINK1.OVL, and MLINK2.OVL. The two overlay files must be on the default drive. To execute MLINK, a command line of the following form is entered:

MLINK file1 [file2 file3 ... filek]

where file1, file2, ... and filek are names of REL, ERL, and/or IRL files. These must be fully qualified file names. If an extension is not specified for a file, then the .REL extension is assumed. MLINK generates two files on the same drive as file1: file1.COM and file1.SYM. The former contains executable object code; the latter contains a symbol table. If an alternative name or drive is desired for the COM and SYM files, the -O option can be used:

MLINK -Oaltfile file1 [file2 file3 ... filek]

where altfile is the alternative file name. If no extension is specified for altfile, the .COM extension is used for the linked output file. The .SYM extension is used for the symbol table.

The ordering of files in the command line is important if one or more of the files is a library of object modules. The files are processed in the order given on the command line. Since MLINK satisfies forward references, an external reference must be unsatisfied when MLINK searches the library containing the object module for that external. In other words, the library file, containing an external that is referenced in another file, should follow that other file in the command line.

MLINK performs selective linking with a library file when the library file name is prefaced with -L. Only those object modules in the library file that are required to satisfy undefined externals will be linked. For instance,

MLINK file1 -Lfile2

generates the linked file1.COM, which does not include any object modules from file2 that are not referenced in file1.

Several optional arguments can be included on the command line:

-Ahhhh where **hhhh** is a hex number that specifies the address where the linker's output file is to be loaded. When this optional argument is omitted a load address of 100 hex is used.

-C This option causes a "compact" output file to be produced by first assigning all initialized program, common, and data segments. It then assigns all uninitialized program, common and data segments. Uninitialized segments are not written to the output file. This option may cause improper operation of code produced by certain compilers.

-E This option must be specified when linking C' programs. It inserts ?CRT0 (C' runtime startup module) into the symbol table as an unsatisfied external. This option must be specified before LIBC is searched.

-Esymbol where **symbol** consists of from one to six characters. This option has the effect of inserting the indicated symbol name into the symbol table as an undefined external.

-Esymbol=hhhh where **symbol** consists of from one to six characters and **hhhh** is a hex address. This option defines the indicated **symbol** to be an entry point with address **hhhh.**

-Iincfile where **incfile** is the name of a file containing one or more MLINK arguments. All arguments from **incfile** are included in the command line in place of the -I argument. The contents of **incfile** are free form and can extend over many lines. MLINK arguments in **incfile** are separated by blanks. Any line in **incfile** that begins with a semi-colon (;) is ignored by MLINK. The -I argument can be nested within any **incfile.**

-Sstring where **string** is a sequence of characters. This option suppresses the writing of any symbol (to the symbol table file) that begins with a character in the indicated **string.** For instance, -S$? suppresses the writing of any symbols beginning with ? or $. If no **string** is specified, a new symbol table file will not be created.

-Uhhhh where **hhhh** is the address given to all unsatisfied externals at the end of pass 1. If **hhhh** is omitted, each unsatisfied external is given a unique address outside any areas assigned by the linker.

-V This is a toggle that causes MLINK to alternate between verbose mode and terse mode. Verbose mode causes MLINK to display the nature of its

processing as it executes. The first use of a -V option turns on the verbose mode, the second -V turns it off, the third -V turns it on, etc.

-X This option suppresses the automatic generation of a jump to the transfer address that may occur at the beginning of the output file.

-Xhhhh where hhhh is a hexadecimal number. This option defines the transfer address (the address at which program execution begins: ?XFER?) to be hhhh. If this option appears before modules with the transfer address, then hhhh overrides the transfer address specified in these modules and MLINK will issue a multiple ?XFER? definition diagnostic.

-XS This option causes the linker to generate stack pointer initialization code, so that the stack pointer will be set to the top of user address space before the start of execution. See section I-E-5 of this manual.

-XShhhh where hhhh is a hexadecimal number. This option both defines the transfer address (?XFER?) to be hhhh and causes stack pointer initialization code to be generated. If this option appears before modules with the transfer address, then hhhh overrides the transfer address specified in these modules and MLINK will issue a multiple ?XFER? definition diagnostic.

-Ystfile where **stfile** is a ZSID type symbol table file. This option enters the symbols from **stfile** into MLINK's symbol table.

-Z This option initializes (to 0) all uninitialized areas on the output file generated by MLINK.

These optional arguments can be used in any order. For instance:

```
MLINK -V -A0200 -OC:PRG PRG -LMDBSL -U
```

C. Using MLINK: Interactive Mode

To execute MLINK in interactive mode, a command line of the following form is entered:

```
MLINK
```

The following response is displayed:

```
Interactive mode: enter MLINK arguments.
To start pass 2, enter a null line.
>
```

The > symbol is the interactive mode prompt. Any argument that can be used in regular mode can also be used in interactive mode. However, since interactive mode is normally verbose, the first -V will turn off

the display of verbose messages, the second -V causes verbose messages to be displayed again, etc. To exit the interactive mode of MLINK, press the carriage return in response to an interactive mode prompt.

D. MLINK Operation

MLINK is a two-pass linker that processes the command line in a left to right fashion. The first pass constructs the symbol table, containing entries for each of three types of segments: program segments, common segments, and data segments. MLINK determines the size of each segment and assigns an absolute address to each. When determining the size of a segment, the largest definition of that segment is used (multiple definitions of common segments are legal). Program segment addresses are assigned first, common segments are then assigned (blank common, if it is used, is the first of these common segments), and data segment addresses are assigned last. MLINK then passes through the symbol table to resolve all entry points, by transforming offsets into absolute addresses. The second pass examines needed files and object modules to generate executable object code.

At any time during MLINK execution (batch or interactive), MLINK can be aborted by typing Control-C or Escape.

E. Miscellaneous Notes for Advanced MLINK Usage

1. The value of ?NEXT? is the first free memory address (in hex).

2. The value of ?XFER? is the address (in hex) where execution of the linked program will begin.

3. If .END. is an unsatisfied external reference at the end of pass 1, then it is set to ?NEXT?.

4. A program being linked can define three entry points (?MEMRY, $MEMRY, .NEXT.) which are treated specially by MLINK. At the end of pass 2, MLINK puts the value of ?NEXT? into the word existing at any of the three special entry points.

5. If a transfer address has not been defined, then MLINK looks for ?MAIN or $MAIN which is taken as the transfer address. If neither of these is found, .MAIN. is taken as the transfer address. If transfer is to .MAIN., MLINK assumes that the program expects the stack pointer to be set to the top of the user address space. MLINK sets this by inserting

```
LHLD 6
SPHL
```

   before jumping (if necessary) to .MAIN.

6. For IRL files, MLINK skips the IRL header.

II. MLINK MESSAGES

A. Normal Operation Messages

The following messages can appear in the course of normal MLINK operations:

MDBS MLINK V1.xx
(C) COPYRIGHT 1982, Micro Data Base Systems, Inc.
Lafayette, IN 47902

Appearance of this message identifies the version of MLINK and indicates that MLINK has successfully begun to execute.

Xfer=hhhh

Execution of the linked program will begin at address hhhh (hex). This message appears only when a transfer address has been specified in a relocatable file.

Next=hhhh

The first free memory address (following those assigned to program, common and data segments) is hhhh (hex). This message indicates the end of MLINK's first pass.

Linking Complete

This message indicates that the linker has finished without encountering errors which would abort the link. Some non-fatal errors may have been encountered.

B. Verbose Mode Messages

When the -V option is used to toggle verbose mode, the following kinds of verbose mode messages may be displayed in addition to normal messages described above.

Code=hhhh

The start of the area that holds all modules' program segments is hhhh (hex).

Common=hhhh

The start of the area that holds all modules' common segments is hhhh (hex).

Data=hhhh

The start of the area that holds all modules' data segments is hhhh (hex).

ffff:

The name of a file (ffff) and all modules selected from ffff are displayed. For each file this display occurs once during the first pass scan and once during the second pass link.

Initialized:
Uninitialized:

These messages precede the Code, Common, and Data messages for their respective areas. See the -C argument in Section I-B of this manual.

nn Modules:

nn (decimal) modules have been selected for linking from the specified file.

nnK table space used:

nn (decimal) K bytes has been dynamically allocated by MLINK for buffers and tables.

C. MLINK Error Messages

If an error occurred in a particular file (rather than in the command line), then the error message is prefaced by the name of that file:

ffff:

If such an error occurred in a REL, ERL or IRL file, then the affected module's name (if known) is displayed in square brackets after the file name and before the message itself.

ffff: [mmmm:]

The possible error messages are shown below. Note that some errors are fatal, while others are non-fatal. Processing is aborted and the user is returned to command mode in the event of a fatal error. Processing continues for non-fatal errors. When processing halts, the message

nn Errors

appears, where nn is the number of error messages MLINK issues during the link.

-A: bad address non-fatal

The A argument did not contain a hexadecimal number or the number contained a bad digit. The argument is ignored and linking continues.

Address outside seg-desc segment non-fatal

A byte outside of the segment currently being worked on was initialized or there is a reference to an address outside of the segment. The segment is indicated by seg-desc which is either: code, data, or a common block name enclosed between / and /. Generally, this condition is caused by incorrect assembler or compiler operation (or by a corrupt REL file) and it typically produces unpredictable results. However, in the case of a memory reference for setting the stack pointer, this condition is reasonable.

Bad argument -z non-fatal

-z is not a valid MLINK argument and is ignored. Processing continues. See section I-B of this manual for a description of valid MLINK arguments.

Bad header non-fatal

The header on an IRL file was not of proper format. This usually indicates a corrupt IRL file. On CP/M, this may have been caused by copying the file with PIP without using the O option of PIP. This file is ignored and linking continues.

Cannot load address xxxx non-fatal

The specified address xxxx is lower than the origin of the output file. This load is ignored and linking continues.

Can't create output file fatal

Under CP/M, this is generally caused by a full disk directory. In rare instances, this is caused by insufficient memory.

Can't create symbol table file fatal

Under CP/M, this is generally caused by a full disk directory. In rare instances, this is caused by insufficient memory.

Can't open ffff non-fatal (pass 1) fatal (pass 2)

The specified file ffff could not be read. This results from a non-existent file, an error in the file name, or in rare instances, insufficient memory (this occurs when input buffer space cannot be allocated). This file is skipped and linking continues (during pass 1).

Chain too long fatal

An external or address chain exceeded 1000 items. This indicates a corrupt REL file or more than 1000 references to a given external within a particular module.

Code size redefined non-fatal

More than one code size declaration was encountered in the specified module, generally a corrupt REL file. The largest definition is used and linking continues.

Common block cccc size mismatch non-fatal

A definition of the specified common block cccc is not as large as the largest definition encountered for that common block. MLINK uses the largest.

Data size redefined non-fatal

More than one data size declaration was encountered in the specified module, generally a corrupt REL file. The largest definition is used and linking continues.

-E: name required non-fatal

A symbol name did not immediately follow the E argument. The argument is ignored and linking continues.

End of file unexpected non-fatal (pass 1)
fatal (pass 2)

The end-of-file marker occurred within module mmmm on file ffff. This is usually an indication that this file has been corrupted. On CP/M, this may have been caused by copying the file with PIP without using the O option of PIP. This module is ignored and linking continues (during pass 1).

***** FATAL ERROR ***** fatal

MLINK is aborting upon encountering a fatal error.

Insufficient memory fatal

MLINK requires more memory to construct all tables required by this link.

***** INTERRUPTED ***** fatal

MLINK is aborting because the Control-C key or Escape key was pressed.

Multiple definition - ssss non-fatal

Entry point ssss is multiply defined. The first definition is used and linking continues.

Multiple definition - ?XFER? non-fatal

More than one transfer address definition was encountered. The first definition is used and linking continues.

nn Unsatisfied externals non-fatal

No entry point definitions were found for the specified externals (a list of names follows this message). This is usually caused by missing or misnamed subroutines, or by out-of-order libraries. If the -U argument was not used, MLINK automatically enters the interactive mode, displaying the message

Type ^C to abort, or CR to start pass 2,
or enter further MLINK arguments now:
>

Typing Control-C aborts processing. Pressing the carriage return causes all unsatisfied externals to be given an address of zero and linking continues. Therefore, during execution, calling such a routine will result in a call to address zero (under CP/M or MP/M this causes the program to exit).

No common block selected - ssss non-fatal

An attempt was made to define an entry point ssss into a common block without first selecting the common block. This generally indicates a corrupt REL file. "Absolute" mode is used for this reference and linking continues.

No modules linked fatal

The link directives did not specify linking of any modules which could be properly accessed by the linker. This is usually due to the fact that one or more of the files specified could not be opened (in this case this message would be preceeded by error messages specifying those files which could not be opened). The linker aborts without creating a new output or symbol file.

No output file name fatal

No output file name was specified by the directives. If this occurs, the O option must be used to specify a name.

Read error non-fatal (pass-1)
fatal (pass-2)

This indicates that the specified file is corrupt. This file is skipped and linking continues (during pass 1).

Read error on output file fatal

An error occurred when attempting to "page-in" a part of the output file. This should not occur.

Unrecognized item - type=d non-fatal

Items of the type indicated by the decimal number d are ignored by MLINK and linking continues.

Write error on output file fatal

Under CP/M, this is generally caused by a full disk.

Write error on symbol table file fatal

Under CP/M, this is generally caused by a full disk.

MDBS MLINK MANUAL

APPENDIX A

RELOCATABLE OBJECT FILE FORMAT

Appendix A

Microsoft REL is a format for RELocatable object files, usually used for 8080/Z80 object code under CP/M. It is a bit-stream, which means that the linker directives (here called items) occur on bit boundaries, rather than byte or word boundaries (except for a couple of special items mentioned later).

Relocatable Object File Format:

Each item in the bit stream consists of a header bit(s), and depending on the item type, other fields (expressed here in BNF grammar):

```
byte := bit bit bit bit bit bit bit bit /* 8 bits */
word := byte byte /* low byte first */
/* address field */
a-field := 0 0 word |          /* absolute */
        0 1 word |      /* code-relative */
        1 0 word |      /* data-relative */
        1 1 word        /* common-relative */
/* name field */
b-field := 0 0 0 |      /* header bits are name char count */
        0 0 1 byte |
        0 1 0 byte byte |
        0 1 1 byte byte byte |
        1 0 0 byte byte byte byte |
        1 0 1 byte byte byte byte byte |
        1 1 0 byte byte byte byte byte byte |
        1 1 1 byte byte byte byte byte byte byte
filler :=  (null) | bit       /* enough bits to force next item */
        bit bit |        /* to byte boundary */
        bit bit bit |
        bit bit bit bit |
        bit bit bit bit bit |
        bit bit bit bit bit bit |
        bit bit bit bit bit bit bit
```

Given all this, the definitions are:

```
REL file            := L_ENDF | module-group L_ENDF

module-group             := module | module module-group

module                   := L_EPRG | item-group L_EPRG

item-group               := relitem | relitem item-group

relitem                  := L_ESYM | L_SCOM | L_PNAM |
                       L_LSCH | L_DCOM | L_CEXT |
                       L_DENT | L_MOFF | L_POFF |
                       L_DSIZ | L_SLOC | L_CADR |
                       L_PSIZ | L_EPRG | L_ENDF |
                       L_WORD | L_BYTE
```


L_ESYM "entry symbol" := 1 0 0 0 0 0 0 b-field

(b-field is name of entry point contained in this module. This item is used during library searches to determine whether this module satisfies an unsatisfied external.)

L_SCOM "select common block" := 1 0 0 0 0 0 1 b-field

(b-field is the name of a common block to use for common-relative references after this item.)

L_PNAM "module name" := 1 0 0 0 0 1 0 b-field

(b-field is the name used by LIB to identify the module. Note: P stands for program, but this really means module.)

L_LSCH "library search file" := 1 0 0 0 0 1 1 b-field

(b-field is the name of a REL file to be searched after all the linker directives are processed if there are still unsatisfied externals.)

L_DCOM "declare common block" := 1 0 0 0 1 0 1 a-field b-field

(b-field is the name of a common block whose size is specified by the a-field (the mode is ignored).)

L_CEXT "chain external" := 1 0 0 0 1 1 0 a-field b-field

(a-field is the head address of the chain, and the b-field is the name of the entry point whose address is to replace into each member of the chain (chain is terminated by an absolute zero entry.)

L_DENT "declare entry point" := 1 0 0 0 1 1 1 a-field b-field

(b-field is the name of an entry point, and a-field is its address.)

L_MOFF "External - offset" := 1 0 0 1 0 0 0 a-field

(a-field is offset to be subtracted from value at this address, after all chaining has been done.)

L_POFF "External + offset" := 1 0 0 1 0 0 1 a-field

(a-field is just like in L_MOFF, but added instead of subtracted.)


```
L_DSIZ "Data area size"        := 1 0 0 1 0 1 0 a-field
```

(a-field is data area size, mode is ignored.)

```
L_SLOC "Set loading location" := 1 0 0 1 0 1 1 a-field
```

(a-field specified the location of the next byte/word is to loaded.)

```
L_CADR "Chain address"         := 1 0 0 1 1 0 0 a-field
```

(a-field is starting address of chain, and present loading location is to replace into each member of the chain (chain is terminated by an absolute zero entry).)

```
L_PSIZ "Module size"           := 1 0 0 1 1 0 1 a-field
```

(a-field is module code area size, mode is ignored.)

```
L_EPRG "Module end, with transfer address"
                               := 1 0 0 1 1 1 0 a-field filler
```

(a-field is the transfer address; absolute zero means no transfer address.)

```
L_ENDF "End of file"           := 1 0 0 1 1 1 1

L_BYTE "Load a byte"           := 0 byte

L_WORD "Load a word"           := 1 0 1 word /* code-relative */
```

In a RELITEM, "ri_type" is the item type, "ri_val" is the a-field (address field) if the item has one, "ri_mode" is the address mode of the a-field, and "ri_name" is a null-terminated array containing the b-field (name field) if the item has one. For the cases of L_BYTE and L_WORD, "ri_val" is the byte/word value, with "ri_mode" the addressing mode for L_WORD.

