Phoenix Software Associates

PLIB86

Object
Library Manager
for Intel 8086/8088

by Dave Hirschman

Table of Contents

Plib86 (tm) is a Phoenix Software Associates Ltd. software system that can manipulate libraries of object files. It supplements the PSA linkage editor Plink86 (tm), and is intended for use on the Intel Corporation (1) 8Ø86 (or 8Ø88) processor (tm) under the MS-DOS (2) or CP/M-86 (3) operating systems.

Plib86 handles object files and libraries conforming to the format generated by the Microsoft compilers for the Intel 8Ø86. This is actually the standard Intel format with an enhanced library index. A list of compilers that produce object code compatible with this format is given later.

The first section of this manual provides an explanation of the "object library" concept and the capabilities of Plib86. User's unfamiliar with library managers would do well to start here. Also, the Plink86 user's guide contains a chapter discussing object files and linkage editors that may be helpful.

The next section of this manual describes how to use Plib86 to handle several common object library situations. At the same time it provides an informal explanation of what the commands do. Those readers experienced with linkage editors and library managers may wish to skip directly to this portion of the manual: it provides enough information to handle most applications.

The final portion of the manual is an exhaustive list of the commands and features offered by Plib86. This should be examined when it becomes necessary to go beyond the examples given in the previous section. Side issues such as error codes are generally referred to appendices.

Trademark Acknowledgements:

(1) INTEL is a trademark of Intel Corporation

(2) MS-DOS is a trademark of Microsoft, Inc.

(3) CP/M-86 is a trademark of Digital Research.

Plib86 also provides a powerful cross-reference function. It optionally generates a report listing each public symbol, the module which defines it, and a list of other modules that refer to it. This may be used to cross-reference a single library or several libraries together, or, in combination with the library search feature described above, to generate a cross-reference of a program that will be created by the linkage editor.

Creating/Merging Libraries

To create a new library use the BUILD command and the FILE command. For example, executing Plib86 and entering

```
BUILD  DB.LIB
FILE   BTREE, SORT, REPGEN,
       FIRSTLIB.LIB;
```

in response to the prompt would create a library named DB.LIB containing the files listed after the FILE command. These files could be single object modules or complete libraries. Everything is merged into a single library.

Normally you can just execute Plib86 and type in commands on as many lines as desired. Then end the last line with a semi-colon to begin processing. Each statement begins with a key word like BUILD or FILE and is followed by arguments, possibly separated by commas. Input is free format, and blank lines are ignored. Also, key words may be abbreviated by leaving off characters at the end. For example, you can use BU and FI instead of BUILD and FILE. An error message will be given if the abbreviation could be confused with another command.

Another way to use Plib8 is to give the commands as it is executed. For example, the above library could have been created by entering (on one line):

```
PLIB86 BU DB FI BTREE, SORT,
       REPGEN, FIRSTLIB.LIB
```

Note that the output file type defaults to LIB automatically.

Library Search

Suppose you want to create a library consisting of several modules plus those portions of another library that are referenced by the modules. Use the LIBRARY command:

```
BU DB FI BTREE, SORT, REPGEN
LIB FIRSTLIB.LIB
```

The portions of FIRSTLIB not referenced by the three other files are not put into the DB library.

Updating a library

To update a library it is necessary to copy the old library to the output file while omitting the module to be updated, and also include the new module. For example to replace module COSINE in library MATHLIB, rename the current MATHLIB.LIB to MATHLIB.OLD and enter

```
BU MATHLIB FI COSINE,
   MATHLIB.OLD EXC COSINE
```

The EXCLUDE statement applies to the previous file name given and causes the COSINE module in the MATHLIB to be ignored.

Module Extraction

The EXTRACT statement causes a single object module file to be created. It may not be used at the same time as BUILD. The first object module found in the input files is extracted, so the particular module to be selected from a library must be specified. The object file extracted may be given any file name. The module name remains the same. For example, typing

```
EXT OLDCOS FI MATHLIB.LIB
      INCLUDE COSINE
```

creates file OLDCOS.OBJ containing object module COSINE. The INCLUDE statement is the counterpart of EXCLUDE: it applies to the previous input file and causes only those modules named to be considered for processing. There wouldn't be any point to INCLUDing more than one module in this case since only the first one found is extracted.

Cross reference listing

To create a cross-reference listing use the LIST command with the same input file commands like those given in previous examples. For example,

```
LIST = DB  S FI BTREE, SORT, REPGEN,
FIRSTLIB.LIB
```

creates a cross reference report named DB.LST describing the modules in all of the files listed. The "S" selects the cross-reference report. For a description of other reports available see the LIST command description. The "=" specifies that the report is to be put into a disk file. If omitted the report appears on the console.

Input Format

This portion of the manual describes some basic input elements. Later sections show how these are combined to create full statements.

Identifiers

An identifier is the name of some object, such as a module or symbol. An identifier is a sequence of no more than 64 characters containing no spaces, and containing none of the following:

```
^=;<>/,\!'#&*+-:@ DEL
```

Lower case letters, when used, are automatically translated into upper case. The first character of an identifier may not be a digit Ø - 9.

The above restrictions on valid identifier characters may be avoided by using the escape character `^`. The character immediately following the escape character is treated as a normal identifier character.

The following are examples of valid identifiers:

```
ProgramI
SORT3
ABC^@       (the `@` is escaped)
```

The following are not valid identifiers:

```
34ABC    - begins with a number
NIM A    - contains a space
```

```
PROG%1  - starts a comment with `%`
```

The above identifiers could all be made valid with the escape character:

```
^34ABC
NIM^ A
PROG^%1
```

To include the escape character in an identifier enter two escape characters `^^`.

Identifiers appearing in object files are truncated to 50 characters for for purposes of comparison with other identifiers in the program. Identifiers may be truncated again for inclusion in reports (see the LIST command).

Disk File Names

Plib86 adapts itself to the file name format used by the operating system it is executing under. The first character not allowed to be in a file name terminates the name. The escape character may be used to put any character into a file name.

In this manual, MS-DOS format file names are used for purposes of discussion. These file names are of the form [device:]name[.type], with optional portions in brackets. Here are some examples:

```
MATHLIB.LIB
B:CHESS.OBJ
SCANNER
```

When the "device:" is not given, Plib86 assumes that the currently logged-in disk is to be used.

Initiating Plib86

Plib86 may be used interactively, or input may be given as it is executed:

```
        Plib86 <statements> <cr>
```

where <cr> means to press the RETURN key. This means that Plib86 may used in .BAT files.

To use Plib86 in the interactive mode, enter

```
                Plib86<cr>
```

on the console. Plib86 will read statements from the console, prompting with "=>". All input is stored uninspected until a carriage return is typed. The standard line editing features supplied by the operating system are available.

A disk file containing all or only part of a command may be inserted into the input at any point by preceding the disk file name with an "@". The default file type is ".LNK". These disk files can contain further "@" specifications, up to three levels deep. The most common use of this feature is to prepare a file containing a complete command; then,

```
        Plib86 @<file name> <cr>
```

creates the library. Sometimes these ".LNK" files may be prepared once for a given library and used over and over again, greatly simplifying the whole process.

Plib86 reads an entire command, checking for syntax only, before any file processing is done.

Command Format

All Plib86 input is free format. Blank lines are ignored, and a command may extend to any number of lines. Comments may be included with input from any source by using a percent sign "%". When this is encountered, all remaining characters on the same line are ignored.

Input is a list of statements:

```
<statement> <statement> ... <statement>
```

Each statement begins with a key word, and many are followed by arguments separated by commas. For example, in

```
FILE A,B,C
```

FILE is the key word, and A, B, and C are the arguments. Key words may be abbreviated by omitting trailing characters, as long as the abbreviation is unique among the entire group of key words. For instance, the previous statement could have been entered as

```
FI A,B,C
```

If a syntax error is found, the current input line is echoed with two question marks inserted after the point at which the error was detected. This is followed by an error message (see Appendix). The command must then be re-entered.

If some other error occurs, Plib86 terminates with an error message also listed in the appendix.

Object Files

Plib86 must be told what object files and libraries to use for input and what modules to select from them. The FILE command is typically used, and normally causes all modules with the given files to be processed:

```
     FILE COSINE, SIN, ARCTAN
```

The LIBRARY and SEARCH commands are similar, but are used only on libraries and select only those modules that define a public symbol that is needed by some other module that has already been processed. This is called a "library search" and is a process carried out by most linkage editors. It insures that only those library modules that are actually needed are included in the program.

```
     LIBRARY MATHLIB
     SEARCH FORTRAN
```

The LIBRARY command causes the given libraries to be searched once. When the SEARCH command is used the libraries may be searched multiple times as long as undefined symbols remain. This won't be needed unless two or more libraries are being searched that each refer to symbols defined in the others.

If Plib86 can't find a requested object file it will look on drive A to find it, and will then ask the operator to enter the drive id. Diskettes may be changed at this time if necessary. Of course, the operator must insure that any diskettes removed do not contain open files like the BUILD or EXTRACT file.

Also, if Plib86 runs out of memory a work file is opened on the default disk, which then may not be removed.

Under MSDOS 2.0 operating systems Plib86 will accept a path name as part of an object file name. Also, if an object file can't be found Plib86 will look for a string named "OBJ" in the environment and append its value to the front of the file name, after stripping any drive id. For example, suppose that the operator enters

```
SET OBJ = \OBJECT
```

and then runs Plib86. Let us suppose that one of the commands to Plib86 is

```
FILE B:TEST.OBJ
```

and that TEST.OBJ doesn't in fact exist on drive B. Plib86 would strip the B: from the name and then try \OBJECT\TEST.OBJ to obtain the requested file.

If an object file (not a library) is being processed the module it contains is given the same module name as the name of the file it came from. This is done because some compilers don't supply a unique module name. This default may be changed by using the AS statement. It supplies the module name for the most recent FILE given. For example,

```
FILE MATH1 AS COSINE
```

would name the module in MATH1 COSINE instead of MATH1.

If you are processing libraries built with Microsoft's library manager you will get several checksum errors. These arise because the Microsoft library manager renames the modules as Plib86 does but does not re-compute the checksum field at the end of the module name record. The messages should no longer appear once the library has been re-built by Plib86.

The modules selected from a library may be further restricted by using the INCLUDE and EXCLUDE statements. These are followed by a list of module names:

```
FILE MATHLIB INCLUDE SIN, COSINE
LIB MATHLIB, DB EXCLUDE BTREE
```

The INCLUDE statement causes only those modules listed to be considered for processing, and this selection precedes a library search. EXCLUDE is the opposite. The modules listed are not processed. INCLUDE and EXCLUDE apply to the FILE, LIBRARY or SEARCH file immediately preceding. In the second example above, for instance, the EXCLUDE BTREE applies only to the DB library, not MATHLIB.

Building a Library

The BUILD command is used to create a library out of the modules selected from the input files. It is followed by the name of the file to create. The file type defaults to .LIB:

```
     BUILD DB.LIB
     BUILD D:MATHLIB
```

After all modules are output the library index is created.

One must be careful that the output file does not have the same name as any of the input files. For instance, entering

```
     BUILD MATHLIB
     FI COSINE, ARCTAN, MATHLIB
```

won't work because MATHLIB will be erased before it is read.

The BUILD command may not be used simultaneously with the EXTRACT command (described next). If no output is requested from Plib86 (i.e there is no BUILD, EXTRACT or LIST command) then Plib86 will simply read the input modules and report any errors it finds.

Extracting a Library Module

The EXTRACT command is used to extract a single object module from a library file and place it into a separate disk file. It is followed by the name of the file to create:

```
EXTRACT COSINE.OBJ
EXTRACT ARCTAN
```

If the file type is omitted OBJ is assumed.

The EXTRACT command extracts the first module found in the input files. Therefore it is usually necessary to use the INCLUDE statement to specify which library module should be extracted. For instance,

```
EXTRACT COSINE FI MATHLIB
```

extracts the very first module in MATHLIB, even if it is not the COSINE module. To get the correct one enter

```
EXTRACT COSINE FI MATHLIB INC COSINE
```

Generating Reports

The LIST command is used to generate reports about the object files being processed. It may optionally be followed by a file name, causing the reports to be directed to that disk file or device. The file name must be preceded by an equal sign. Then a character is entered for each report desired, separated by commas. There are currently two reports available:

M - A list of all modules processed in alphabetical order.

S - A list of all public and external symbols in alphabetical order. Each is followed by the name of the module defining the symbol in parenthesis (this will be blank for external symbols). Following this is an alphabetical list of all modules that access the symbol (i.e. this is a cross-reference report).

Here are some examples:

```
LIST M
LIST = DB.LST M, S
LIST = XREF.LST S
```

The report generator can be re-configured for different size paper. It assumes 8Ø columns and 66 rows per page as a default. The number of columns may be changed with the WIDTH command, and the number of rows with the HEIGHT command. Here are some examples:

```
     WIDTH 132
     HEIGHT 88
```

Controlling the Library Index

Normally all public symbols from all modules are inserted into the library index. If a duplicate symbol is found library creation continues but a warning message is given and the index entry for that symbol will select the first module defining the symbol.

Sometimes it is useful to exclude certain symbols from the library index. This may be accomplished by using the NOINDEX command. For example,

```
NOINDEX SYM1, SYM2, SYM3
```

excludes SYM1, SYM2, and SYM3 from the index.

Suppose you wish to create a library that contains several versions of the same module, for instance a device driver for some kind of hardware. If you try to place all of the modules into the library you will get duplicate symbol warnings, and at link time the linkage editor wouldn't be able to select the desired module.

This can be made to work by using NOINDEX on the module entry points. This exlcudes all of these symbols from the library index. To get the linkage editor to select the correct modules insert an un-used but unique dummy symbol into each module. At linkage edit time one of these dummy symbols would be accessed in order to create a need for the desired module. The linkage editor would then select it when the library is searched.

Using Plink86, for instance, one could use a statement like

```
DEFINE FOO=DRIVER1
```

to select the module containing driver 1. An alternative is to rely on the fact that the name of each module is actually in the library index as well, followed by an exclamation point. For example, if the library contains a module named DRIVER1 then there will be a dummy index entry named DRIVER1!. These symbols can be used instead of creating a dummy module entry point as discussed above.

Most of the common error conditions detected by Plib86 result in a console message that should be self-explanatory. For the more uncommon or obscure errors a number is printed on the console that may be looked up below.

Command Syntax Errors

The following errors are caused by mistakes made in the input given to Plib86. The input line causing the problem will be displayed on the terminal, with a couple of question marks inserted at the point where the error was detected. Re-run Plib86 after correcting the problem.

1 - "@" files are nested too deeply. Only three levels of "@" files may be active at any given time. Do you have a loop in your "@" file references?

2 - Disk error encountered while reading "@" file. Try re-building the file.

5 - The item given for input at this point is too large. The maximum size allowed is 64 characters.

6 - Invalid digit in number (i.e. not Ø thru 9).

1Ø - Invalid file name. The input stream should contain a valid file name for the particular operating system being used.

11 - Expecting a statement. A key word which begins a statement should be present here.

12 - The INCLUDE and EXCLUDE statements may not be used simultaneously on the same input file.

14 - Expecting identifier. A section, module, segment, or symbol name must be entered at this point.

15 - Expecting "="

16 - Expecting a value. An expression or 16-bit quantity must appear at this point.

17 - No files were given to process! You must use the FILE statement and specify at least one input file.

18 - The BUILD and EXTRACT commands may not be used simultaneously. You must run Plib86 twice with one command in each.

Work File Errors

When Plib86 runs out of memory it opens a work file on disk named Plib86.WRK to hold the description of the library. These error codes indicate a problem with processing the work file.

30 - The work file can't be created. Probably there is no space in the disk directory.

31 - An I/O error occurred while writing the work file.

32 - An I/O error occurred while reading the work file.

33 - An I/O error occurred while positioning the work file.

34 - There are too many module description objects in this library (about 50,000 symbols, modules, and so on may be defined). This library is too large for Plib86 to handle.

Input Object File Errors

The following errors have to do with the object files that are given to Plib86 to process. Usually they occur when a file has been corrupted somehow. Try re-compiling to get a new copy of the object file. If it is a library supplied by the compiler manufacturer that is causing the problem, try to get a fresh copy of it.

41 - Premature end of input object file. The end of the indicated file was reached unexpectedly. Possibly, the file was truncated by copying it with a program that assumes a CNTL-Z (1AH) is end of file.

42 - Fatal read error in object input file.

43 - Fatal file position error in object input file. This can occur when a library file is truncated.

Output File Errors

The following errors are caused by a problem in creating the output code file or memory map file (when written to disk). Often, they are caused by a full disk or disk directory, a disk that is write-protected, or some kind of hardware problem with the disk.

45 - Can't create output disk file. Possibly the disk directory is full, or the disk is write protected.

46 - Output file too large. The given modules won't fit into the library. You will have to break up the library into one or more smaller ones.

47 - Fatal disk write error in output file. Possibly the disk is full or write protected, or some kind of hardware error has occurred.

48 - Fatal disk read error in output file. Probably, an irrecoverable hardware error has occurred.

49 - Can't close output file. Probably the disk is write protected, or a hardware error has occurred.

5Ø - Can't create the LIST output file. Possibly the disk directory is full, or the disk is write protected.

Miscellaneous Errors

51 - There are too many symbols to be placed into the library index. You will have to break up the library into one or more smaller ones.

52 - No modules were selected (by library search, INCLUDE, or EXCLUDE) to be placed in the output file (BUILD or EXTRACT).

54 - There isn't enough memory in the computer to run Plib86. You must have a really tiny memory - better buy more!

Plib86 Bugs

These errors indicate a bug in Plib86 has occurred through no fault of your own. They are listed here for completeness in the manual, although it is unlikely that you can do anything to correct them. Try running Plib86 again. If the error persists, please gather the relevant information and contact the software distributor from whom you obtained Plib86.

201 - Expandable array bug.

205 - Seek errors while writing output file (attempt to seek past end of file).

219 - Bad object block (GetBlock).

221 - Invalid object key (Q).

222 - Invalid object key (QM).