TLIB Transputer Librarian User Guide

TLIB Version 91.1
11/15/91

Contents

Introduction

Overview

TLIB is a librarian for use with the TLNK Transputer linker. It allows the creation, modification and examination of libraries of relocatable files. TLIB operates in a fashion reminiscent of a line oriented text editor with commands to append, insert, delete, and otherwise manipulate libraries.

System Requirements

TLNK is modest in its requirements for memory. It should run in any environment which supports other major system development tools (compilers, etc). TASM uses a single temporary file which is the same order of size as the library being operated on.

Usage

Getting Started

The general form of the TLIB command line is:

```
tlib <library_filename> [-options]* [option_specific]*
```

The "library_filename" field MUST contain the name of the library being mangled (with an explicit filename extension if other than "`.tll`"). After this field the syntax depends on the operation being requested.

One feature common to all the operations is the use of an optional "`-v`" command line flag to enable or disable the verbose output mode. On Macintosh versions of TLIB the "-v" flag ENABLES the verbose output mode. Under other operating systems the "-v" flag DISABLES the verbose output mode. In the following TLIB description and examples the assumption is made that "-v" will DISABLE the verbose output mode; if you are using TLIB on a Mac you will need to remember to mentally reverse this assumption.

Note that in general one type of operation may be done to the library for each invocation of TLIB. You may append multiple files at a time for example, but if you also wish to delete a file you must do it in a separate operation.

Since TLNK supports a single pass library scan algorithm, order will usually be important in a library created with TLIB. TLIB disallows multiple copies of the same relocatable file (keyed by the name), but you may have different files in the library which define the same external symbols. Your libraries should be structured such that the TLNK linker only gets one defining instance of each external symbol (otherwise a fatal link error will result). TLIB can be instructed to read a series of files to be placed in a

library and automatically determine a correct ordering (see the "build" option description in the **Examples** section).

Note that during operations which require modification of the library a temporary file is used whose name is formed be replacing the filename extension of the library (usually ".tll"), with ".1". This file is created on the current directory. The use of this file limits the time window during which a system crash could potentially corrupt the library to be between when the library is actually deleted and the temporary file is renamed to replace it.

If the "library_filename" you specify does not exist, TLIB will create a skeleton version for you (and give you a warning message), prior to doing whatever you have requested to the library.

Note that full pathnames may be used for the library and relocatable files used with TLIB. A point to watch about doing this with input relocatable files is that the full pathname must be used from then on to refer to them in the library and they will be written back to wherever the pathname specifies if they are ever removed from the library.

The executable version of TLIB being shipped allows for command line "wildcard" expansion for filenames (such as "`*.trl`"). This feature is of utility for making a library with the "build" option and also is handy during library "update" operations (assuming you have a directory full of the relocatable files to use). This feature may or may not be available with other "C" compilers and operating systems.

Examples

Assume you have a library named "`zip.tll`" which you wish to examine. The syntax would be:

```
tlib zip
```

This would give you a verbose listing which would show which files the library contains and which external symbols are defined in each file. If you just wanted a listing of which files were in the library:

```
tlib zip -v
```

Now suppose that you want to add two files (named "`zap.trl`" and "`zow.trl`"), to the end of the library. This looks like:

```
tlib zip -a zap zow
```

This will append the two files in the order given. You may explicitly specify filename extensions if the default of "`.trl`" is not wanted. If you don't like TLIB to tell you what is being done you could give the "`-v`" option:

t`lib zip -av zap zow` or perhaps `tlib zip -a zap zow -v`

Instead of appending to the end of the library, you may insert files in a specific position in the library. This is done by specifying a "key" file as well as one or more files to be inserted. The files are inserted immediately BEFORE the "key" file in the order they are given on the command line. For example, if you wished to insert the files "`hi.trl`" and "`mom.trl`" between the "`zap.trl`" and "`zow.trl`" files added in the previous example:

```
tlib zip -i zow hi mom
```

Again (as always), you may use the "`-v`" switch to turn off the blow-by-blow commentary. Another common operation done to libraries is to "update" specific files within the library while leaving the majority of files untouched. This is accomplished using the TLIB "`-u`" command. Assuming you have a newer version of "`hi.trl`" and "`mom.trl`" that you wish to substitute into the library, the following will do the job:

```
tlib zip -u hi mom
```

For the special case of building a library from scratch, TLIB offers a "build" command as an alternative to simply appending the desired files (which does work). The "build" command offers the additional feature of doing a topological sort of the symbol dependencies in each file and then appending the files onto the library such that a single pass over the library will satisfy all external references whose definitions are also present in the library. Assuming that you wish to create a new library by the name of "`zip.tll`" and you wish to place three files in it ("`zap`","`zow`" and "`mom`"):

```
tlib zip -b zap zow mom
```

One limitation of the "build" option is that it requires that the files being placed in the library contain NO cycles in the external symbol dependency graph. If this assumption is false you will get an error message informing you of the fact and listing one of the files involved in the cycle.

A less common operation which TLIB supports is making a "copy" of files in the library. This is occasionally handy when building a new "special" library out of several existing libraries. Suppose you want a copy of "`zip.trl`", "`hi.trl`" and "`zow.trl`":

```
tlib zip -c zip hi zow
```

Note that the files being copied need not be listed in the order they appear in the library. The copy operation creates the requested individual files, it does not affect the library itself in any way. Another point: TLIB will set the file "modification" time and date of the library file copy to be the same as that of the original file before it was added to the library. The modification times and dates of all files in a library are remembered to allow this. Updating of the modification time and date is also done for all other TLIB operations which make individual copies of files from libraries.

The above commands were mostly concerned with expanding or updating the contents of a library. The following commands are used to remove files from libraries. As a starter, if you wish to completely dissolve a library (named "zip.trl"), into its component files:

```
tlib zip -s
```

This will "split" a library into a bunch of individual files. It is often convenient to perform this operation on an empty directory since if the library contained a significant number of files the directory gets somewhat cluttered. Assuming you wish to maintain the library, but want to remove several files you have two options: You may remove specified files and make individual copies of them ("extract" them), or you may simply "delete" them. The following example shows the "extract" operation:

```
tlib zip -e hi mom
```

In this case "`hi.trl`" and "`mom.trl`" are removed from library "`zip.tll`" and turned into individual files. As usual you may use the "`-v`" option flag to disable the verbose output messages. The syntax to do the corresponding "delete" operation is:

```
tlib zip -d hi mom
```

Note that in the case of both "delete" and "extract" you do not have to specify the desired files in the same order as they appear in the library.

Option Descriptions

The above examples cover most of the operational details of TLIB. Remember that the library filename must be the first item on the command line after the program name (TLIB), and that only one "command" may be used per execution of TLIB. The relocatable filenames and option flags may be mixed in an arbitrary fashion on the command line; order is only important with filenames which are being appended or inserted into a library. Finally, you may use the "`-v`" switch to disable verbose output.

Note that TLIB provides a return value of non-zero to the operating system if errors are detected. This may be used by some shell programs to respond to the error in a programmable fashion. If an error was detected an error message is written to standard output regardless of whether the "`-v`" option was specified.

The following command listing summarizes the information presented in the examples:

```
library_name [-v]
```

List the names of files in "`library_name`" and the external symbols defined within them (assuming no "`-v`" option), to standard output. The "modification" time and date associated with each file is also listed.

```
library_name -a [-v] append_filename [append_filename]*
```

Append one or more files to the end of "`library_name`" in the order specified. Nothing is written to standard output if the "`-v`" option is given.

```
library_name -b [-v] build_filename [build_filename]*
```

Create a library using the supplied files. The files are placed in the library in an order which satisfies the external reference dependency graph of the files (ensures that one pass over the library is adequate). If a cycle exists in the graph, and this ordering can't be done, you will get a error message indicating one of the files involved in the cycle. At this point you should either remove one of the offending files, or perhaps split the offending file into two or more pieces partitioning the external definitions and references such that the dependency cycle is removed. When doing this you may have to create multiple copies of some of the definitions such that all the external references in the revised library can be satisfied in one pass. In practice these problems arise infrequently, and they may be avoided by simply having TLNK search the same library more than once. Nothing is written to standard output if the "`-v`" option is given.

`library_name -c [-v] copy_filename [copy_filename]*`

Copy the specified files from "`library_name`" into individual files. Nothing is written to standard output if the "`-v`" option is given.

`library_name -d [-v] delete_filename [delete_filename]*`

Delete the specified files from "`library_name`". Nothing is written to standard output if the "`-v`" option is given.

`library_name -e [-v] extract_filename [extract_filename]*`

Remove the specified files from "`library_name`" and put into individual files. Nothing is written to standard output if the "`-v`" option is given.

`library_name -i [-v] insert_before_filename insert_filename [insert_filename]*`

Insert the specified files immediately before the "`insert_before_filename`" in "`library_name`". The order the files are inserted in reflects the order in which they appear on the command line. Nothing is written to standard output if the "`-v`" option is given.

`library_name -s [-v]`

Split "`library_name`" into individual files. The "`library_name`" file is deleted. Nothing is written to standard output if the "`-v`" option is given.

`library_name -u [-v] update_filename [update_filename]*`

Replace the specified files in "library_name" with the contents of the respective individual files. Nothing is written to standard output if the "`-v`" option is given. If any of the files to be updated aren't already in the library they are appended to the library (in order), after all other library files.

Appendix A: Error Messages

Types of Error Messages

There are two classes of error messages which TLIB can generate:

1. **Warnings**. These are used to report problems which aren't severe enough to cause TLIB to exit with a non-zero return value. These messages usually indicate trouble which isn't immediate, but may be soon! The message format is:

```
WARNING: message_text
```

2. **Fatal errors**. If the problem detected by TLIB is so severe that it can't continue operating, it will give a "fatal" error message:

```
FATAL: message_text
```

After printing one of these messages, TLIB will immediately exit with its error return code set (non-zero).

Error Message Descriptions

The following descriptions list the various error messages which TLIB can generate (in alphabetic order). Note that error messages which have a trailing "`filename`" will display the name of the actual file involved:

`FATAL: Bad record format on file: filename`

This error message is generated whenever a record in an input or library file doesn't conform to a legal record type. This indicates that the indicated file has been corrupted, or that a change made to the record format generated by TASM or TLNK has not been made to TLIB (for those who can't help doing a little "improving" to programs now and then).

`FATAL: Conflicting options selected`

You can only do one type of operation on a library per execution of TLIB. You may not mix operations (such as delete a file and add two more), except as separate steps. You may delete several files or add several files in one step however. Another possible source of this is errors in the command line to TLIB which cause it to interpret a filename as commands (no filenames with leading "-" allowed).

`FATAL: Error determining where input file pointer is: filename`

This happens when TLIB gets a error return when attempting to determine.the specifed input file pointer position via a "`ftell`". This generally indicates file system trouble.

`FATAL: Error determining where library file pointer is: filename`

This happens when TLIB gets a error return when attempting to determine the library file pointer position via a "`ftell`". This generally indicates file system trouble.

`FATAL: Error determining where temp file pointer is: filename`

This happens when TLIB gets a error return when attempting to determine the temporary file pointer position via a "`ftell`". This generally indicates file system trouble.

`FATAL: Error reading input file: filename`

This indicates that TLIB got a error return when reading a file to be appended to, or insert in, a library. Generally this happens because of a premature EOF in a corrupted file.

`FATAL: Error reading library file: filename`

This indicates that TLIB got an error return when reading the specified library file. Generally this happens because of a premature EOF in a corrupted file.

`FATAL: Error seeking input file: filename`

This happens when TLIB gets a error return when attempting to do a "`fseek`" on the specified input file. This generally indicates file system trouble.

`FATAL: Error seeking library file: filename`

This happens when TLIB gets a error return when attempting to do a "`fseek`" on the library file. This generally indicates file system trouble.

`FATAL: Error seeking temp file: filename`

This happens when TLIB gets a error return when attempting to do a "`fseek`" on the library temp file. This generally indicates file system trouble.

`FATAL: Error writing output file: filename`

This error is generated when TLIB gets an error return while writing a relocatable output file. This error generally indicates a write-protected directory or insufficient file space.

`FATAL: Error writing library file: filename`

This error is generated when TLIB gets an error return while writing a skeleton library file (given that the one you specified doesn't exist). The causes include a write-protected directory and insufficient file space.

`FATAL: Error writing temp file: filename`

This error happens when TLIB gets an error return while writing to the library temp file. The causes include a write-protected directory and insufficient file space.

`FATAL: External symbol dependency cycle detected involving file: filename`

This error is detected when you are using the library "build" option ("`-b`"), and a cycle is detected in the graph which is created by reading the external symbol definitions and references for ALL the files to be placed in the library. For those who like computer science jargon this indicates that the dependency graph was not a DAG. For ideas on how to get around this problem, see the description of the "`-b`" option in the **Option Descriptions** section.

`FATAL: General usage is 'tlib library_filename [options]*'`

This error message is given in response to an incorrect invocation of TLIB (such as not providing any command line arguments). Following the error message TLIB lists a short description of the various commands it accepts and the desired command line format.

`FATAL: Insufficient filename string memory`

This error message occurs when TLIB gets rebuffed during an attempt to get memory for storing the filenames in a library. This indicates that the "C" heap (via "`malloc`"), was full. Possible solutions to this are to reduce the size of other data structures which share memory with the heap (see `MAX_FILES` and `MAX_DEF` in "tlib.c"). If you can't do this, or don't wish to recompile TLIB, you can split the offending library into two smaller libraries to get around the problem.

`FATAL: Insufficient file dependency node memory`

This error message is only generated when you are constructing a "new" library with the "build" option ("`-b`"). All the "`out-of-memory`" error messages generated during the use of this command should be treated as equivalent (ie. you need more memory or less external symbols in less files), unless you are interested in the fine points of TLIB operation (such as when you are adding a new feature!)

`FATAL: Insufficient symbol reference/definition memory`

This error message is only generated when you are constructing a "new" library with the "build" option ("`-b`"). All the "`out-of-memory`" error messages generated during the use of this command should be treated as equivalent (ie. you need more memory or less external symbols in less files), unless you are interested in the fine points of TLIB operation (such as when you are adding a new feature!)

`FATAL: Insufficient symbol string memory`

This error message occurs when TLIB gets rebuffed during an attempt to get memory for storing the external symbol names in a particular library file. This is needed when the symbol names are being sorted to generate the `T_SYMBOL` library overhead records (see the "TASM/TLNK/TLIB Relocatable Record and File Formats" documentation for a description of the T_SYMBOL record). This indicates that the "C" heap (via "`malloc`"), was full. Possible solutions to this are to reduce the size of other data structures which share memory with the heap (see `MAX_FILES` and `MAX_DEF` in "`tlib.c`"). If you can't do this, or don't wish to recompile TLIB, you must reduce the number of external definitions in the file. Note that the memory is only needed while the

file is being added to the library and that the space is returned and may be reused for other files.

`FATAL: Library already contains file: filename`

This error message occurs when you attempt to add or insert a file into a library which already contains a file by that name.

`FATAL: Library doesn't contain file: filename`

The named file is not present in the library. This happens when TLIB is looking for a file to insert before, or when a series of files is being extracted, deleted or copied out.

`FATAL: Library file name is already in use: filename`

The "filename" was listed as the name of a library to be "built". The "build" option ("`-b`"), may only be used for constructing new libraries.

`FATAL: Specified library file isn't in library format`

The library file you provided for TLIB to use wasn't a library or has been corrupted.

`FATAL: The size of SLONG is not correctly configured`

This error message can only occur if you re-compile TLIB. It indicates that the SLONG "`typedef`" in "`taldef.h`" is set to an integral type of less than 4 bytes length. This type should be signed and at least 4 bytes long to allow TLIB to work correctly.

`FATAL: Too many defined symbols in file: filename`

This error message is generated if a file being appended or inserted into a library contains too many external symbol definitions. As released, TLIB can handle 1000 symbol definitions per file. If desired you may increase this limit by changing the value of `MAX_DEF` at the beginning of "`tlib.c`".

`FATAL: Too many files in library: filename`

This error occurs when you attempt to add or insert a file into a library which is full. As released TLIB can handle 1000 files in a library. This limit is controlled by the `MAX_FILES` macro located at the beginning of "`tlib.c`".

`FATAL: Unable to close input file: filename`

For some reason the specified input file couldn't be closed. This generally indicates a problem with the file system.

`FATAL: Unable to close library file: filename`

This occurs when the library file you specified didn't exist; TLIB then created a skeleton version but was unable to close the newly created file. The file is closed to allow changing the file access to be "read-only" in place of the "write" ability needed during creation of the skeleton file.

`FATAL: Unable to close output file: filename`

This error message occurs when a relocatable file is to be written from information contained in the library and the output file used couldn't be closed. This generally indicates a problem with the file system.

`FATAL: Unable to close update file: filename`

For some reason the specified input update file couldn't be closed. This generally indicates a problem with the file system.

`FATAL: Unable to find end of input file: filename`

This error message indicates that TLIB couldn't find the `T_EOF` (end-of-file), record at the end of an input file. This indicates either that the file has been corrupted, or that the operating system being used doesn't keep track of file lengths to the byte (see the "`FATAL: Unable to find end of library file`" error message description for more information on file length problems).

`FATAL: Unable to find end of library file: filename`

During library manipulations, TLIB attempts to find the T_EOF (end-of file), record at the end of the library. This error message indicates that it was unable to do so. TLIB depends on a file system which keeps a count of exactly how many bytes are present in a file. If your file system only allows EOF to be at a block boundary you will have to make extensive changes to TLIB. All the operating systems for which TLIB is distributed are capable of determining file length accurately. Another possible cause of this message is the library having gotten padded with garbage during translation from another system via a serial I/O or disk conversion technique which alters file length. Finally, it's possible the library file simply got corrupted somehow.

`FATAL: Unable to find end of update file: filename`

This means that TLIB couldn't find the `T_EOF` (end-of-file), record for the file which is being used to update the library. This indicates either that the file has been corrupted, or that the operating system being used doesn't keep track of file lengths to the byte (see the "`FATAL: Unable to find end of library file`" error message description for more information on file length problems).

`FATAL: Unable to get modification time for file: filename`

This happens when TLIB is putting a new file (or new version of an old file), in the library, but a "`fstat`" operation on it got an error return. The "`fstat`" was done to obtain a last modification date/time to store in the library with the file.

`FATAL: Unable to mix code for different CPU types`

Libraries may only contain code and data for a particular class of Transputer (T414B, T800, etc). TLIB uses the information provided in the relocatable files to ensure that all the files in a particular library are for one type of processor.

`FATAL: Unable to open created library file: filename`

This occurs when TLIB couldn't find the library you supplied, made a skeleton library for you and then couldn't open the newly created library. This error message probably indicates problems with your file system.

`FATAL: Unable to open input file: filename`

This error message occurs when the input relocatable file for a append or insert operation couldn't be opened. Usually this means that you provided the wrong file pathname or extension.

`FATAL: Unable to open library temp file: filename`

For operations which require changes to the library, TLIB uses a temp file. This error message occurs when TLIB is unable to open/create it. Note that the temp file is located on the current directory which should be writeable.

`FATAL: Unable to open output file: filename`

This error message happens when TLIB was unable to open/create an output file to write files from a library on. This usually means the desired directory is write-protected, the file system is full, etc. Note that if a pathname was used when a file was inserted into the library the same pathname will be used when detemining where to write the file during a library "split" operation.

`FATAL: Unable to open update file: filename`

TLIB generates this error message when it can't open a relocatable file which is to be used to update the corresponding file in the library. Usually this means that you provided the wrong file pathname or extension.

`FATAL: Unable to open/create library file: filename`

This error message happens when the library file you specified doesn't exist and TLIB was unable to open the file to create it. This can indicate a write-protected or full directory, or file system trouble.

`FATAL: Unable to set modification time for file: filename`

This happens when TLIB is extracting a file from a library and attempting to set the "modification" time of the extracted file. TLIB does this using a call to "`utime`" (whose return value caused this error). The modification time is set to ensure that it matches the modification time present when the file was originally added to the library.

`WARNING: Library doesn't contain file: filename`

The named file is not present in the library. This happens when TLIB has been instructed to "update" a file, but the file is not already in the library. TLIB resolves this by appending the indicated file to the end of the library (after any files which actually could be updated).

`WARNING: No external symbols defined in file: filename`

The named file being added to the library has no external symbol definitions. Although this is ok, it means that TLNK will never use this file since the file can never satisfy any external references. This may still be useful if the library is serving as a transport medium for multiple relocatable files which will be removed later and used in another context.

`WARNING: Unable to change temp file name: filename`

This error is generated by TLIB during the final stage of replacing the original library file with the temp file. If, after the original library has been deleted, the temp file can't be renamed with the same name, this error is generated. At this point the library is intact but is named with an extension of ".1" instead of whatever the original filename extension was.

`WARNING: Unable to close library file: filename`

This is generated when TLIB is closing the original (or newly created), library file and got an error return. This is not fatal since presumably the temp file has the updated contents of the library by now (or the library being "split" and deleted).

`WARNING: Unable to close temp file: filename`

This indicates that TLIB was unable to close the library temp file during cleanup operations. If TLIB had previously detected an error the next step would have been to remove the temp file. If no errors had previously been detected the next steps would have been to delete the old library file and rename the temp file to be the library file. As usual, this error indicates some trouble with the file or operating system.

`WARNING: Unable to open library file, creating: filename`

The library file you provided (as the first command line argument) doesn't exist and TLIB has made a skeleton (empty), version for you. If the library really DOES exist this indicates that you entered the name wrong, provided an incorrect or missing filename extension (the default extension is ".tll"), or the library isn't in the currect directory, etc.

`WARNING: Unable to remove library file: filename`

This is generated when TLIB attempts to remove the library file after a library has been "split" but gets an error return. This generally indicates a write-protected directory or similar problem.

`WARNING: Unable to remove old library file: filename`

This error is detected when TLIB has successfully written a new version of the library to the temp file and is now trying to delete the original library so that the temp file can be renamed with the same name. This generally indicates a write-protected directory or file system problem.

`WARNING: Unable to remove temp file: filename`

This indicates that TLIB got another error while trying to delete the library temp file which was used. Since the temp file was being deleted (instead of replacing the old library file), a previous error must have occurred. The cause will probably be related to whatever caused the earlier error.

Appendix B: TLIB Internals

Source Code Organization and Compiling

TLIB consists of one "C" source file ("tlib.c"), and uses only standard "C" include files and "taldef.h" which is common to the other assembly and interface programs in the Transputer Toolset.

For MS-DOS source distributions the "makefile" file may be used with the supplied MAKE utility to build the "tlib.exe" executable using Microsoft "C" V6.0a or Borland C++ V2.0 (the Microsoft/Borland "C" compilers are not supplied and must be purchased separately). For Macintosh source file distributions consult the supplemental information your vendor has included with the Transputer Toolset.