TCC Command Line Driver User Guide

TCC Version 91.1
12/1/91

Contents

Introduction

Overview

TCC is a program designed to automate the process of compiling, assembling, and linking Transputer programs. Instead of running PP, TCX, TASM and TLNK individually, you can simply use TCC and have it take care of calling the other tools as needed. In addition to reducing the number of commands you must issue to compile a program, TCC also takes care of choosing suitable command line parameters and "intermediate" files for use with the various tools.

TCC may be used to process programs consisting of more than one source file. TCC may also be used together with MAKE to automate the processing of extremely complex programs.

TCC has been designed to be as compatible as possible with the "cc" type programs commonly provided with other compilers. Although differences exist due to the special requirements of the Transputer, those familiar with other "cc" programs will find TCC easy to use.

System Requirements

TCC is modest in its use of memory. It should run in any environment which supports other major system development tools (compilers, etc). Since it does occupy some memory, it is possible that a tool being run from TCC could run out of memory when it wouldn't if run directly from the command line. This shouldn't happen unless the environment is extremely memory restricted, or the files being processed are very large. The only file TCC uses itself is a small "command" file it creates for use with TLNK. TCC will also direct the various tools it calls to do file I/O during the processing of programs.

TCC uses the setting of the "TMP" environment variable to determine which directory to tell the various tools to read and write temporary files to. If the "TMP" variable doesn't exist, the current directory will be used to hold the temporary files. When finished, TCC will remove all the temporary files it created (or caused to be created).

Usage

Getting Started

The general form of the TCC command line is:

```
tcc <input_filename>* [-options]*
```

At least one "input_filename" must be provided. The "extension" of the input file name will determine what type of processing TCC will perform. Zero or more "options" may be specified to control the operation of TCC in finer detail.

Examples

Assume you have a program named "exam1.c" that you wish to compile, assemble and link. The syntax would be:

```
tcc exam1.c
```

Two output files will be written: "exam1.trl" which is the relocatable object file written by TASM, and "exam1.tld" which is the linked executable produced by TLNK. In the above case (and by default), programs are compiled and linked with the T414 Transputer as the target processor. If you are compiling for the T800, the syntax would be:

```
tcc exam1.c -t8
```

By default, TCC will echo the command lines it issues to the various tools it calls. You can tell TCC to be "quiet" by providing a "-v" option switch. For example:

```
tcc exam1.c -t8 -v
```

TCC allows you to group options together behind the same "-" option lead-in, as long as no option other than the last requires a parameter. Although the "t" option does require a parameter ("8"), the "v" option doesn't, so it can be placed before the "t" option:

```
tcc exam1.c -vt8
```

This command line is completely equivalent to the previous one.

Also note that what the "-v" option really does is "toggle" whether TCC is verbose or not. Depending on how TCC is configured when it was created for a particular host environment, TCC could be "quiet" by default and issuing a "-v" could make TCC verbose. This is generally the case for versions of TCC running on the Macintosh, for example. For the purposes of this documentation we will assume that TCC is verbose by default and that "-v" makes TCC be "quiet". If this isn't the case for your version of TCC, please remember to mentally "flip" the meaning!

Of course, TCC may also be used to process other types of files in addition to "C" files. For example, to assemble and link an assembly language source file named "zip.tal":

```
tcc zip.tal
```

Two output files will be written: "zip.trl" which is the relocatable object file written by TASM, and "zip.tld" which is the linked executable. As in the previous cases with "C" source files, this file will be assumed to be targeted for a T414 Transputer. If you wished to assemble the program for the T800 instead:

```
tcc zip.tal -t8
```

One of the nice feature of the **Transputer Toolset** is that the TASM assembler has been designed to support the use of the PP "C" preprocessor for optional include file processing, macro substitution, etc. TCC assumes that files with the extension ".pal" are assembly language source files which require preprocessing with PP. For example:

```
tcc hello.pal -t8
```

This will take the "hello.pal" file and first run it through PP before assembling and linking it. Many of the assembly language example programs provided with the **Transputer Toolset** are of this type.

The final type of file TCC recognizes is a relocatable object file. These files are generally written by TASM and have a file name extension of ".trl". To reduce the length of the TCC command line, TCC assumes that any listed file without an extension really has an extension of ".trl", and needs to be linked. Thus, the following two commands have the same meaning:

```
tcc zap
tcc zap.trl
```

As before, the "zap.trl" file will be linked with the T414 runtime library to produce a "zap.tld" file. To link "zap.trl" for the T800, you could use:

```
tcc zap.trl -t8
```

Of course, for correct operation the "zap.trl" file must have been previously compiled/assembled for the same target processor!

As mentioned, you may use TCC to combine more than one input file into a single resulting executable. These input files may also be of different types. For example:

```
tcc test1.c test2.pal test3.tal test4
```

This will cause TCC to run PP/TCX/TASM on test.c to produce a "test1.trl", run PP/TASM on "test2.pal" to produce a "test2.trl" and run TASM on "test3.tal" to produce a "test3.trl". TCC will then combine "test1.trl", "test2.trl", "test3.trl" and "test4.trl" into a single executable file named "test1.tld". By default TCC uses the first input file name as the name of the resulting executable file. If you wished to override this and name the resulting executable file "test.tld", you could use:

```
tcc test1.c test2.pal test3.tal test4 -o test.tld
```

The file name argument to the "-o" option can also be listed directly after the "-o" flag:

```
tcc test1.c test2.pal test3.tal test4 -otest.tld
```

TCC Environment Variable

The next section of this documentation goes into some detail describing the various options which TCC supports. Depending on your use of TCC you may find yourself "always" using certain options. For example, if you are only using T800 type transputers, you might get tired of always specifying "-t8" on the command line. You can avoid this by creating an environment variable named "TCC" to which you can assign whatever options you customarily issue to TCC. From the perspective of TCC, the string assigned to the environment variable (if it exists), is processed first, followed by the actual command line.

The "-v" option is another good candidate for use with this environment variable: If your taste in tools is the strong "silent" type, just add a "-v" to the "TCC" variable value!

Option Descriptions

The above examples pretty well described how input files are dealt with. This section describes the many options which are supported to customize the operation of TCC.

One issue not mentioned in the previous discussion is what happens when you get an error! If any of the tools TCC is using returns a non-zero exit code, then TCC removes whatever temporary files it has created and calls "exit" itself, with the same error code. If TCC itself finds an error it returns an exist code of one. If everything works fine then TCC returns an exit code of zero.

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

`-a address`

The "`address`" field specifies the initial load address for the linked executable file. The contents of the field will be shipped verbatim to TLNK and should be a "C" style decimal, hexadecimal or octal constant. If no "-a" option is provided the load address will default to the first off-chip memory location. Use of this option should be coordinated with the "-w" option described later. See the TLNK manual for more information about choosing load and stack addresses.

`-c`

When this option is present TCC will not link any relocatable object files it generates (or is passed). No executable file will be produced.

`-d macro[=value]`

Tells TCC to pass PP the specified macro definition. This option is useful with both "C" programs and with assembly language programs which are being preprocessed by PP (".pal" input file name extensions).

`-e entrypoint`

The "`entrypoint`" field specifies where the linked executable file should be started. The contents of the field will be shipped verbatim to TLNK and should either be a "C" style constant, or the name of a global symbol defined within the program (or runtime library). If no "-e" option is provided, the entrypoint symbol will default to "_main". See the TLNK manual for more information about choosing an entry point.

`-g`

This option tells TCC you wish to include debug information in the files being processed. Doing so will facilitate later use with debugging and performance analysis tools at the cost of a slightly larger and slower executable program.

`-i pathname`

Tells TCC to pass PP the specified include file search path. This option is useful with both "C" programs and with assembly language programs which are being preprocessed by PP (".pal" input file name extensions).

`-l pathname`

Provide a user library ("`pathname`"), to be searched by TLNK before it searches the standard runtime library.

`-m pathname`

Specifies that TCC should have TLNK produce a linkage listing (sometimes called a "map"), written to the specified file. This "map" will document which files were linked together, indicate where the various data structures and functions will be loaded in memory, etc. Note that no file name extension will be provided by default, whatever you specify will be what you get. The suggested file name extension for this type of file is ".map". See the TLNK documentation for detailed information about the listing file format and contents.

`-o pathname`

By default, TCC uses the first source file name as a template in creating the linked output file name (a file name extension of ".tld" is substituted for whatever the source file had). Using this option you can override the default behavior and force any output file name you wish. Note that no file name extension will be added to the file name you provide; whatever you specify will be what you get. The suggested file name extension for this type of file is ".tld".

`-p`

This option tells TCC you only wish to preprocess the input source files. This option is useful with both ".c" and ".pal" input file types. The preprocessed output files will have the same name as the original input files except the file name extension will be ".pp".

`-s`

This option tells TCC you wish to preprocess and compile the input "C" source files, but not assemble or link the results. The preprocessed/compiled output files will have the same name as the original input files except the file name extension will be ".tal".

```
-t [f]type
```

This option tells TCC what type of processor and floating point model you wish to generate code for. The optional "`f`" character tells TCC you wish to have all floating point operations done using IEEE 32 bit format. The default is to use IEEE 32 bit math for "float" types and IEEE 64 bit math for "double" types.

The following Transputer processors are allowed as code generation targets in the "`type`" field:

t0	Any 32 bit CPU
t2	T212, T222
t25	T225
t4	T414
t45	T400, T425
t8	T800
t85	T801, T805

To compile code for the T225 CPU, you would use:

```
-t25
```

Or, equivalently:

```
-t 25
```

If you wish to compile code for the T800 with all floating point math done using IEEE 32 bit math:

```
-t f8
```

And so on. For more information about this option see the "-p" and "-f" options in the TCX documentation.

```
-u macro
```

Tells TCC to pass PP the specified macro un-definition. This option is useful with both "C" programs and with assembly language programs which are being preprocessed by PP (".pal" input file name extensions).

```
-v
```

Toggle TCC verbose output status. Since TCC will also pass this option to the various tools it uses, this will effect everything done by this TCC call.

```
-w address
```

The "`address`" field specifies the initial workspace (stack), address for the linked executable file. The contents of the field will be shipped verbatim to TLNK and should be a "C" style decimal, hexadecimal or octal constant. If no "-w" option is provided the stack address will default to the last on-chip memory location. Use of this option should be coordinated with the "-a" option described earlier. See the TLNK manual for more information about choosing load and stack addresses.

```
+a argument
```

This is a "meta" option to TCC. TCC takes the provided "`argument`" string and passes it without interpretation to TASM. This option lets you control TASM options which have no explicit TCC option. For example:

```
+a -l
```

The TASM "`-l`" option causes TASM to generate an assembly language listing file. Note that you must also specify the "-g" TCC option so that the debugging information needed by TASM to make a listing is preserved. See the TASM documentation for information about what options are available.

```
+c argument
```

This is a "meta" option to TCC. TCC takes the provided "`argument`" string and passes it without interpretation to TCX. This option lets you control TCX options which have no explicit TCC option. For example:

```
+c -i
```

The TCX "`-i`" option causes TCX to use a temp file to cache compiler intermediate information. This option allows the compilation of larger source files on memory-limited host systems at a cost of somewhat slower compilation. See the TCX documentation for information about what options are available.

```
+l argument
```

This is a "meta" option to TCC. TCC takes the provided "`argument`" string and passes it without interpretation to TLNK as the argument for the "Flags" TLNK command file directive. This option lets you specify TLNK options which have no explicit TCC option. For example:

```
+l a
```

This tells TLNK that you wish to have the symbol table listing in the linkage map sorted by address rather than the default alphabetic ordering. You should also specify the "-m" TCC option in order to generate the map in the first place. See the TLNK documentation for information about what options are available.

```
+p argument
```

This is a "meta" option to TCC. TCC takes the provided "`argument`" string and passes it without interpretation to PP. This option lets you control PP options which have no explicit TCC option. For example:

```
+p -c5
```

The PP "`-c5`" option causes PP to recognize and process incoming ANSI Trigraph character sequences. Note that since the "`argument`" must be a single "item" from the perspective of TCC, the following is NOT an equivalent invocation:

```
+p -c 5
```

TCC will pass a "`-c`" through to PP and then try to interpret the "`5`" as a TCC option.

See the PP documentation for information about what options are available.

Appendix A: Error Messages

Types of Error Messages

TCC only generates "fatal" errors. These have the following format:

```
FATAL: message_text
```

After printing one of these messages, TCC will immediately exit with an error code of one.

Error Message Descriptions

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

```
FATAL: -C flag may not be used with -O, -P, or -S flags
```

You can't tell TCC to not link the files it processes while at the same time providing one or more of the "`-O`", "`-P`", or "`-S`" options. Note that options assigned to the TCC environment variable are considered the same as those provided on the command line for the sake of this error check.

```
FATAL: Duplicate -C flag
```

Only one "-C" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Duplicate -G flag
```

Only one "-G" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Duplicate -P flag
```

Only one "-P" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Duplicate -S flag
```

Only one "-S" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Duplicate -T flag
```

Only one "-T" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Duplicate -V flag
```

Only one "-V" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Error writing TLNK command file: filename
```

This error is generated when TCC gets an error return while writing the command file for TLNK. This error generally indicates a write-protected directory or insufficient file space.

```
FATAL: Filename too long
```

One of the input file path names provided to TCC was longer than the configuration limit defined by FNSIZE in "taldef.h". This limit is generally at least 80 characters.

```
FATAL: Illegal Transputer type
```

TCC produces this error message when it encounters an illegal argument for the "-t" option.

```
FATAL: Insufficient argument memory
```

This error message occurs when TCC gets rebuffed during an attempt to get memory for storing information about a command line, or "TCC" environment variable, argument. The possible solutions include anything which will increase the amount of available memory TCC has to use.

```
FATAL: Insufficient 'argv' memory
```

This error message occurs when TCC is unable to "malloc" memory for the "argv" structure to be passed to either PP, TCX, TASM or TLNK. The possible solutions include anything which will increase the amount of available memory TCC has to use.

`FATAL: Insufficient environment copy memory`

This error message occurs when TCC gets rebuffed during an attempt to get memory for a copy of the "TCC" environment variable. This is not very likely to happen in practice, but if it wasn't checked for the probability would surely rise!

`FATAL: Invalid '+' flag: argument`

The argument to a '+' "meta" option wasn't one that TCC recognized. The "`argument`" field details the unrecognized argument.

`FATAL: Missing argument for flag: option_flag`

The TCC option listed in the "`option_flag`" field requires an argument which TCC couldn't find.

`FATAL: -O flag may not be used with -C, -P, or -S flags`

You can't provide TCC with an explicit TLNK output file name while at the same time providing one or more of the "`-C`", "`-P`", or "`-S`" options. Note that options assigned to the TCC environment variable are considered the same as those provided on the command line for the sake of this error check.

`FATAL: Only one initial workspace is permitted`

Only one "-W" flag may be specified. This includes options assigned to the TCC environment variable.

`FATAL: Only one load address is permitted`

Only one "-A" flag may be specified. This includes options assigned to the TCC environment variable.

`FATAL: Only one output file is permitted`

Only one "-O" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Only one program entrypoint is permitted
```

Only one "-E" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: Only one TLNK map file is permitted
```

Only one "-M" flag may be specified. This includes options assigned to the TCC environment variable.

```
FATAL: -P flag may not be used with -C, -O, or -S flags
```

You can't tell TCC to only preprocess the input files while at the same time providing one or more of the "`-C`", "`-O`", or "`-S`" options. Note that options assigned to the TCC environment variable are considered the same as those provided on the command line for the sake of this error check.

```
FATAL: Temporary filename too long
```

This indicates that the length of one of the temporary file names TCC creates is longer than allowed by the FNSIZE configuration macro in "taldef.h". This limit is generally at least 80 characters. This error probably indicates that the directory path assigned to the "TMP" environment variable is too long, since it forms a preamble to the temporary file name (which is itself fairly short).

```
FATAL: Too many temp files
```

This error should be impossible for TCC to generate unless it has been modified. It indicates that the pool of distinct temporary file names TCC uses internally is exhausted. If you get this error, and haven't been playing with the TCC source code, please record all the information about the situation that you feel is relevant and forward the information to Logical Systems (or the dealer where you purchased the software).

```
FATAL: Unable to close TLNK command file: filename
```

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

```
FATAL: Unable to open TLNK command file: filename
```

Unless told not to link with the "-c" option, TCC will create a temporary "command" file to give to TLNK. This file will be located in the temporary directory specified by the "TMP" environment variable, if it exists. If not, it will be written to the current directory. This error might indicate a error in the setting for "TMP", or a write-protected directory, etc.

Appendix B: TCC Internals

Source Code Organization and Compiling

TCC consists of one "C" source file ("tcc.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 "tcc.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.