tracecc and trana, version 1.3.7

Dipl.-Ing. D. Krause

November 26, 2003

Contents

1 Overview

Tracecc is a debugging preprocessor for C, C++, Objective-C and Java. It processes .ctr, .cpt, .mtr and .jtr files to produce .c, .cpp, .m or .java files.
The input files contain tracing instructions in a special notation. Depending on the options given to tracecc the tracing instructions are converted into C code (when producing debug versions) or removed (when producing a shipping version).
When a debug version of a program is run it produces output either to standard output or to a debug file.
The trana program converts the plain text debug file (as written by the program) into a beautified version. This beautified version can be opened by editors like gvim[1] which are capable of folding.

2 Special instructions for tracecc

2.1 How to write tracecc special instructions

2.1.1 Notation for special instructions

- `$(`*instruction arguments*`)`
  The special instruction is finished by the closing bracket.
  Here an example:
  ```
  $(trace-init myfile.deb)
  ```
  The function *trace-init* is invoked with argument *myfile.deb*.

- `$!`*instruction arguments*
  The special instruction is finished by the end of line.
  Example:
  ```
  $!trace-init myfile.deb
  ```

Both notations are equivalent.

2.1.2 Output special instruction

- `$?` *format arguments*
  The output special instruction is finished by the end of line, *format* and *arguments* are used as in *printf()*- or *fprintf()*-calls.
  ```
  $?  "x is %d", x
  ```

[1] http://www.vim.org

2.2 Adding the header file for trace functions

Use

```
$(trace-include)
```

to insert an include-line for the trace functions.
This instruction should be placed after the last "normal" include line.
When running tracecc for debugging it is replaced by

```
#include <dktrace.h>
```

otherwise by an empty line.

2.3 Starting and finishing debug output

The

```
$(trace-init filename)
```

produces code to start trace output.
Depending on the tracecc options trace output goes to the specified filename or to standard output.
The

```
$(trace-end)
```

instruction produces code to finish trace output.

2.4 Trace output

The

```
$? format arguments
```

instruction produces trace output.

Example:

```
int my_square(int i)
{
  int back = 0;
  $? "+ my_square i=%d ", i
  back = i * i;
  $? "- my_square back=%d", back
  return back;
}
```

The *format* string in the lines for entering a function (and only in these lines) should have a plus sign, a space and the function name in the beginning. This is needed by the trana program.
The *format* string in the lines for returning from a function (and only in these lines) should have a minus sign, a space and the function name at the beginning.
Format strings for lines indicating an error condition should have an exclamation mark at the beginning.
Format strings for all lines indicating normal program flow should have a dot at the beginning.
Example:

```
double my_square_root(double d)
{
  double back = 0.0;
  $? "+ my_square_root %lf", d
  if(d >= 0) {
    back = sqrt(d);
    $? ". argument %lf ok", d
  } else {
    $? "! wrong argument %lf", d
  }
  $? "- my_square_root %lf", back
  return back;
}
```

2.5 Other special instructions

2.5.1 Turning debug mode temporarily on / off

- `$(trace-off)`
  `$(debug-off)`
  turns debugging temporarily off.

- `$(trace-on)`
  `$(debug-on)`
  turns debugging back on.

- `$(timestamp-on)`
  turns timestamps in trace messages on.

- `$(timestamp-off)`
  turns timestamps in trace messages off.

These options only take affect if tracecc is run generally in debug mode.
The purpose is to disable trace messages for special code sections (i.e. functions) to decrease output file size.

2.5.2 Turn line numbers on / off

- `$(lineno-on)`
  turns line number printing on.

- `$(lineno-off)`
  turns line number printing off.

2.5.3 Reading a text file to an array of strings

- `$(string-file` *filename*`)`
  reads the given file and converts it into an array of string pointers finished by a `NULL` pointer.

2.5.4 Reading a file into an array of bytes

- `$(byte-file` *filename*`)`
  reads the given file and converts it into an array of unsigned chars.

2.5.5 Reading output of a command

- `$(pipe` *command*`)`
  `$(|` *command*`)`
  runs the given command and inserts the output of the command execution at the current place in the file.

2.5.6 Adding code in debug mode only

- `$(trace-code` *code*`)`
  `$(!` *code*`)`
  inserts the source code at this position only if tracecc is run in debug mode.

3 Running tracecc

3.1 Program invocation

Use

`tracecc`

or

`tracecc` *options*

to run tracecc on the current directory.
Use

`tracecc` *options* *inputfile* *outputfile*

to run tracecc for a given input and output file.
Use

`tracecc` *options* *directory*

to run on a specified directory.

3.2 Options

3.2.1 Getting help

- `-h`
  `--help`
  shows a help text.

- `-v`
  `--version`
  shows version information.

3.2.2 Debugging

- `-d`
  `--debug-enable`
  turns debug mode on.

- `-d-`
  `--debug-enable=no`
  turns debug mode off.

- `-s`
  `--debug-stdout`
  configures trace messages to go to standard output instead of the file.

- `-s-`
  `--debug-stdout=no`
  makes sure that trace messages go to file, not to standard output.
- `-t`
  `--debug-timestamp`
  adds a time stamp to the trace messages.
- `-t-`
  `--debug-timestamp=no`
  removes the time stamp from the trace messages.
- `-k`
  `--keyword`
  adds the "trace" keyword to all trace messages.
- `-k-`
  `--keyword=no`
  removes the trace keyword from the trace messages.

3.2.3 IDE integration

- `-m`
  `--make`
  causes tracecc to check the modification times of files before processing when running on a directory.
- `-m-`
  `--make=no`
  turns modification time check off.
- `-l`
  `--linenumbers`
  adds `#line xx "yyy"` lines to the destination file. This allows to track down compiler errors to the original source.
- `-l-`
  `--linenumbers=no`
  turns line number compiler directives off.
- `-i`
  `--debug-ide`
  configures trace messages to be printed in an IDE style (if debug mode is on).

- `-i-`
  `--debug-ide=no`
  turns IDE style for trace messages off.

3.2.4 Output file style

- `-p`
  `--cpp-comments`
  allows C++ style comments to be passed to the destination file.

- `-p-`
  `--cpp-comments=no`
  converts C++ style comments into normal C comments.

- `-b` *boxwidth*
  `--boxwidth=`*boxwidth*
  sets the box width for comment boxes.

3.3 Command line options and permanent options (preferences)

3.3.1 Overview

Tracecc – as most other dklibs based programs – can be configured to use permanent options in conjunction with the command line options.
If tracecc is run permanent options are processed before the command line options are read.
This has both advantages and disadvantages:

- You can configure options to be permanent, so you do not have to type them for each program invokation.

- You must keep in mind that permanent options might be in use.

The following options can be set permanently:

Table 1: Permanent options overview

Option on	Option off	Purpose
-d	-d-	Debug mode
-s	-s-	Trace messages to standard output
-i	-i none	IDE style for debug messages
-k	-k-	Keyword "trace" in trace messages
		to be continued

		Continuation
-t	-t-	Timestamp in trace messages
-l	-l-	Line number compiler directives
-p	-p-	Allow C++ style comments
-b *width*		Comment box width
-m	-m-	Make style when running on directories

3.3.2 Inspecting permanent options

Use

```
tracecc -C
tracecc --show-configuration
```

to inspect permanent options.

3.3.3 Installing permanent options

Use

`tracecc -c` *options*
`tracecc --configure` *options*

to set permanent options.

3.3.4 Removing permanent options

Use

```
tracecc -u
tracecc --unconfigure
```

to remove all permanent options.

3.3.5 Ignoring permanent options temporarily

Use

`tracecc -r` *options*
`tracecc --reset` *options*

to skip processing of permanent options.
When the `-r` option is processed all options – also those set by command line arguments – are reset to the default values.
If `-r` is used it is strongly recommended to use it as the first argument.

4 Formatting trace output using trana

4.1 Overview

Once you have run

```
tracecc -d
make
```

and executed your program you will have a debug file.
The suggested suffix for debug files is .deb or .tr.
This debug file (or trace messages file) is a plain text file where each line is started by a file name and a line number.
If the -k option was specified when running tracecc the trace keyword occures next.
If the *format* argument was choosen as suggested in section 2.4 on page 4 the next character should be one of the following:

- `+`
  if the line indicates a function invokation.
- `-`
  if the line indicates a return from a function.
- `.`
  if the line is informational.
- `!`
  if the line indicates an error or something critical.

Next there is text consisting of comments and/or variable contents.

This flat file is difficult to read so we want to use the folding capabilities of editors as i.e. VIM.
Before we can do so we must convert the file to prepare it.
VIM can do text folding based on markers (special text placed in the file) or based on the indent level.
Under normal circumstances I suggest to use the indent level.

4.2 Running trana

4.2.1 Command line arguments

The command line syntax for trana is

`trana` *options*
`trana` *options* *inputfile*
`trana` *options* *inputfile* *outputfile*

You can use the following options:

- `-m`
  `--marker`
  adds markers for vim folding to the output file. This is only necessary if you want to have folding based on markers.

- `-t`
  `--trace-only`
  transfers only those lines to the output file which contain the trace keyword. This is only necessary if you wrote trace output to standard output and it is mixed with normal program output. In this case the -k keyword is necessary when running tracecc. In general this is not recommended.

- `-n`
  `--numeric`
  inserts the numeric indent level before each line.

- `-p`
  `--position`
  transfers the source position to the output file.

You can turn of an option by appending a minus to the short form or a "=no" to the long form.
Example

- `-m-`
  `--`marker=no

4.2.2 Command line options and permanent options

The options above can be installed permanently by

`trana -c` *options*

The current permanent options can be viewed by

`trana -C`

To remove all permanent options use

`trana -u`

To skip permanent options for one program invokation use

`trana -r` *options*

The long options

```
--configure
--show-configuration
--unconfigure
--reset
```

can be used instead.

4.2.3 Getting help

- `-h`
  `--help`
  shows a help text.

- `-v`
  `--version`
  shows version information.

5 Inspecting output with Vim

When inspecting output produced by trana you can do folding either by indent level or by markers.
Folding means that a number of lines is hidden, a summary line is shown instead. Opening a fold means the summary line is replaced by the text formerly hidden.

5.1 Folding by indent level

Type the following commands:

```
:setlocal sw=2
:setlocal foldmethod=indent
:setlocal foldopen=all
```

5.2 Folding by markers

Type the following commands:

```
:setlocal foldmethod=marker
:setlocal foldopen=all
```

5.3 Navigating through the folded file

In on-text-mode (not command-mode) use the key sequence `zr` to open all folds one level. Use `zm` to close one fold level.
To open one fold place type the key sequence `zo` after placing the cursor on the summary line. The key sequence `zc` closes one fold.
The

```
:setlocal foldopen=all
```

automaticall opens all the folds you place the cursor in.