LAUTERBACH
DEVELOPMENT TOOLS

Application Note for t32cast

MANUAL

Application Note for t32cast

Application Note for t32cast

Version 06-Jun-2024

History

18-Apr-23	Added commands for source code instrumentation.
11-Jun-18	Initial version.

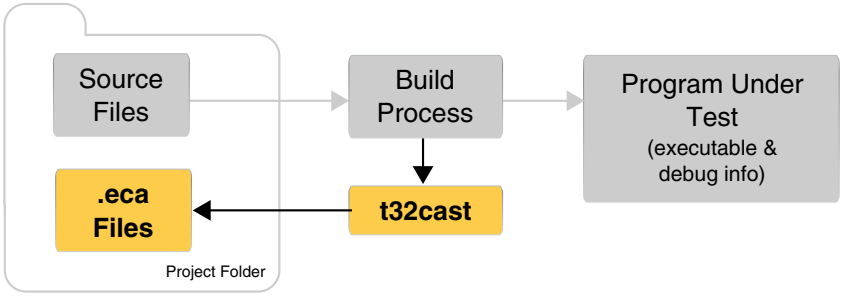
Introduction

There are TRACE32 features which require additional source code details that are not included in the debug infromation generated by the compiler. For example, several coverage metrics such as MC/DC or decision coverage need information about the locations of decisions and function calls in the source code. Furthermore, decisions and conditions may not be visible in the processor flow trace for all architectures and compilers. This required the decisions in the source code to be instrumented with additional code.

The task of the command line tool **t32cast** is to generate all the additional information. It analyzes the C/C++ source code and generates an Extended Code Analysis data file (.eca) per source file. In case of required source code instrumentation, **t32cast** can be instructed to instrument source files where necessary.

The **.eca** files have to be loaded into TRACE32 if needed. The source code file and the .eca file must be located in the same directory. In this way, the assignment is simple and unambiguous.

t32cast can be started via a PRACTICE script or a batch job. But to ensure that the .eca files always match the source code files, it is recommended to integrate their generation into the build process.



Intended Audience

- Developers who want to use one of the **TRACE32 code coverage metrics** that require additional details about the source code.
- Persons in charge of the build system who want to integrate t32cast into their build process.

Prerequisites

Windows users will need to ensure that the Microsoft Visual C++ Redistributable package is installed on their system. The latest one is always available as a free-of-charge download from the Microsoft download center.

Linux users will need at least libncurses-5 or libncurses-compat. Depending upon the distribution used, other dependencies may be needed. Generally, running the **t32cast** tool will highlight any missing components.

Related Documents

- **"Application Note for Trace-Based Code Coverage"** (app_code_coverage.pdf).
- The description of the **sYmbol.ECA** command group in **"General Commands Reference Guide S"** (general_ref_s.pdf).

Restrictions

Currently, **t32cast** does not parse decisions within macros within macros correctly. The source file must be pre-processed by the compiler to expand these before running it against t32cast.

Installation

System Requirements

- 64-bit Windows or Linux host computer.
- The **sYmbol.ECA** command group was supported for the first time with TRACE32 build of 95748 (March 2018). The version of TRACE32 can be verified with the **VERSION.SOFTWARE** command.

License Requirements

The use of **t32cast** requires no additional licenses.

Installing t32cast

The command line tool t32cast is located under **~~/bin/<host_os>**. An installation is not necessary.

If you receive an error message about a missing *.dll, then install the Microsoft Visual C++ Redistributable package. See **"Prerequisites"**, page 5.

Command Line Parameters of t32cast

The command line tool t32cast is designed to be included as part of the build process.

The examples below show how you can quickly display the help message (**--help**) and the version information (**--version**). They are independent of the t32cast integration into the build process.

```
t32cast [subcommand] [options] <input>
```

Available subcommands:

eca	Generate data for extended code analysis in TRACE32. This command generates additional information about the C/C++ code which is usually not included in the debug symbols. The data includes the position of decisions, function calls and comments.
instrument	Instrument source file for MC/DC or Decision coverage measurement with TRACE32. Depending on the target and compiler the source file may be partially or fully instrumented. Refer to **"Application Note for Trace-Based Code Coverage"** (app_code_coverage.pdf) for more information.
vectors	Generate MC/DC test vectors. This command will output all decisions together with test vectors to achieve full MC/DC coverage. The results may be used for developing test cases. They are not for use with **TRACE32**.

The **t32cast** utility may require the output from the compiler's pre-processing step, if the build flags cannot be passed directly to **t32cast** with -I, -D or --extra-arg. Most compilers generate these intermediary files but automatically remove them as part of the normal build process. These files contain the full source code, any included files and all compiler macros have been replaced or expanded. Please refer to your toolchain documentation to configure your compiler to generate and keep these files.

The **t32cast** tool will generate an Extended Code Analysis (ECA) data file. These should be loaded into **TRACE32** to be able to calculate coverage data for several coverage metrics.

Examples for Arm, Power and TriCore Architecture can be found under **~~/demo/t32cast/eca**. These contain a complete build environment showing how to adapt a Makefile for GNU make and the target-specific compiler.

Common Options for All Subcommands

--help	Display the help message and exit.
--version	Display version information and exit.
-D *<macro[=value]>*	Define a macro.
-I *<dir>*	Add directory to include search path.
--extra-arg=*<string>*	Additional argument to append to the **t32cast** parser options. All options available to the Clang compiler are available.
--extra-arg-before=*<string>*	Additional argument to prepend to the **t32cast** parser options. All options available to the Clang compiler are available.
-imacros *<file>*	Define macros from file before parsing.
-m32	Preprocess source code for 32-bit target.
-m64	Preprocess source code for 64-bit target.
-o *<output>*	Write output to file *<output>*.
--show-warnings	Show additional warnings when parsing the C or C++ code
--stop-on-error	Stop on Clang parser errors. This can help diagnosing missing decision in **TRACE32**.
--strict-mode	Abort the parsing process on any warning. This can help diagnosing missing decision in **TRACE32**.
-x *<language>*	Set the language as which the file as parsed. The options are C and C++.

Additional Options for ECA Subcommands

--comments	Include C/C++ comments in output.
--export-cfg	Include control flow information in ECA file which is needed when performing coverage measurements with code instrumentation.

--parse-const-expr	Exports decisions where the expression can already be evaluated during compile-time
--parse-includes	Include decisions in code from included header files

Additional Options for INSTRUMENT Subcommand

--filter <file>	Specify filter file for partial instrumentation
--gen-instr-source-files	Generate instrumentation probes needed for compilation of instrumented source files. By default, the files are saved in the current working directory. The path can be changed with the **--probe-dir** option.
--mode <value>	Specify instrumentation mode. Allowed values: • MCDC
--num-arg-probes=<num>	Used in conjunction with **--gen-instr-source-files.** Needed in cases of functions with many parameters containing decisions.
--probe-dir=<dir>	Specify output directory of instrumentation probe files. When using this option as part of the instrumentation itself and the same directory is provided with each instrumentation call, the number of argument probes will be adjusted automatically. In this case a manual call with the **--gen-instr-source-files** option is not necessary.

Examples

Example 1: This PRACTICE script (*.cmm) shows how to start **t32cast** from within **TRACE32** and display the help message.

```
OS.screen cmd /C C:/T32/bin/windows64/t32cast.exe --help && pause
```

Example 2: This batch script (*.bat) displays the version number of **t32cast** in a shell window.

```
@echo off
C:
cd C:\T32\bin\windows64\
t32cast.exe --version && pause
```

Example 3: Call **t32cast** directly to analyze the file foo.c and create the output file foo.c.eca

```
t32cast.exe eca -o foo.c.eca foo.c
```

Example 4: Call **t32cast** directly to analyze the file foo.c with an additional include path and a defined preprocessor macro

```
t32cast.exe eca -I ./include -D DEBUG=1 -o foo.c.eca foo.c
```

Example 5: Call **t32cast** directly to instrument all decisions in the file foo.c

```
t32cast.exe instrument --mode=mcdc -o foo.instr.c foo.c
```

Example 6: Call **t32cast** directly to instrument all decisions defined by the gaps file gaps.json in foo.c

```
t32cast.exe instrument --mode=mcdc --filter gaps.json -o foo.instr.c
foo.c
```