01 March 2024

Change Documentation for

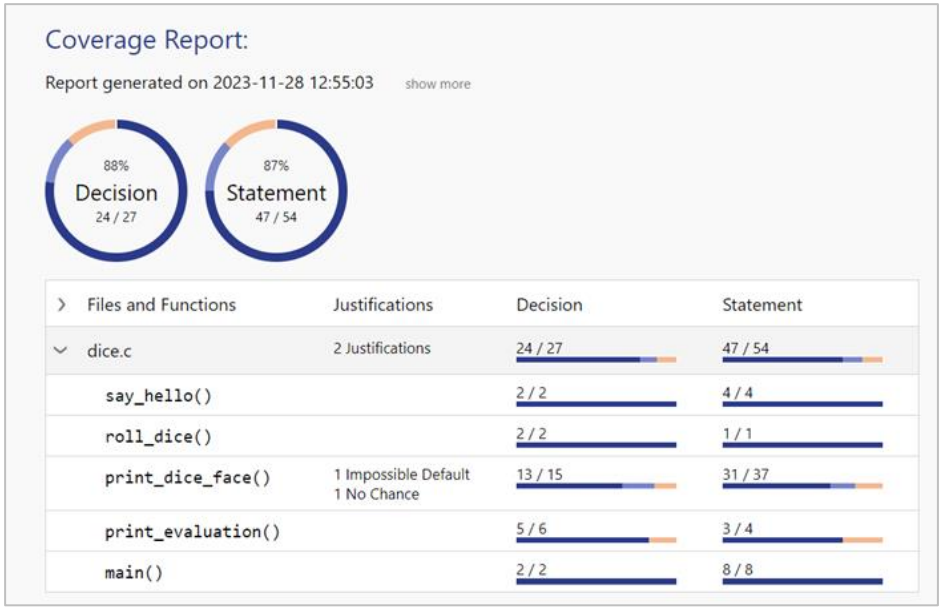
Testwell CTC++

Version 10.1.0

Features and Changes

Justifications

Missing coverage can now be justified, and these justifications transparently influence all coverage measures.



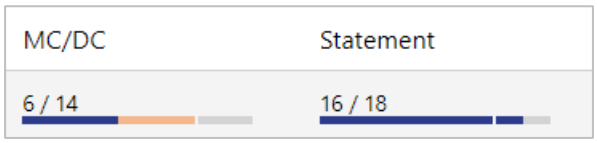
Justifications can be stored in source code comments or in companion files. With a justification, a *tag* used for grouped presentation in overviews and an *explanation* are associated.

Example in source code:

```
// CTC++ Justify False | EFFORT: Simulation too expensive, because…
if (world_still_exists) …
```

Coverage charts

In ring and bar charts of the HTML report, the visualization changes.



- Covered parts are always shown in dark blue.
- If coverage ratio is below threshold (white mark), missing coverage up to the threshold is colored in light red. Missing coverage above threshold is shown in light grey.
- Influence of justifications is shown in a lighter blue.

Single-file text templates

Beside structured HTML reports, also text-based reports can now be generated with **ctcreport**. This report generation is fully based on templates, adaptable by the user.
With these templates, coverage information on project (overall), directory, file, and function level can be reported. Coverage below function level, e.g. for probes, lines etc., is still exclusively reported in source code view of structured templates.
With installation, three single-file templates are delivered in the **ctcreport** folder:

- **example_csv.csv**: coverage per function,
- **example_xml.xml**: overall coverage and hierarchically per directory, file, and function,
- **example_markdown.md**: coverage per source file, grouped by directories.

Templates from arbitrary locations

HTML templates as well as new single-file templates do not need to be in **ctcreport** folder anymore. With option `-template` in **ctcreport** call, any file can be referenced using a file extension:

```
ctcreport -template corporate-design.zip …
```

looks for **corporate-design.zip** (i.e., an HTML template) in current directory.

```
ctcreport -template example_csv …
```

looks in **ctcreport** folder and uses the first template file with this base name, for example **example_csv.csv**.

Origin of instrumented files

Instrumented included files can exist in many copies – one for each including source file in a project. **ctcreport** aggregates these copies into one or more variants due to preprocessing (`#ifdef` etc.).
In source code view of each variant, the including source files leading to this variant are now listed. This list is grouped by symbol file containing either one or more including files or a direct instrumentation of the considered file or file variant.

Reduction of deprecated **ctc** macros

For instrumented builds, several macros are no longer supported by **ctc.h:**
CTC_COND_OPER_NO_INSTR, CTC_DECL_COND_NO_INSTR, CTC_EMIT_RCS_STAMP, CTC_RCS_STAMP, CTC_EMIT_SCCS_STAMP, CTC_SCCS_STAMP.

Unified `const` recognition

For decisions in ternary-? operators, the same recognition for compile-constantness is now used as for other decisions.

Handling of Testwell CTC++ Help from command-line

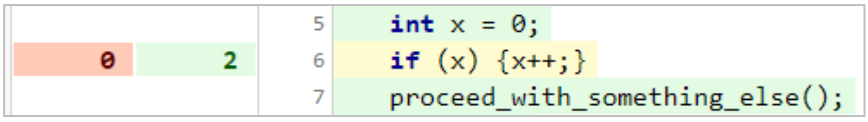
- With new option `-H`, tools **ctc**, **ctcreport**, **ctcpost**, **ctclaunch**, **ctc2dat,** and **ctcxmlmerge** open the Testwell CTC++ Help in standard browser with the tool's description as landing page.
- With `-H word`, the search function is opened with "word" as search term.
- For Linux and macOS, man pages are no longer provided. Existing man pages from a former installation are not deleted, please delete them as they are no longer updated.

Performance Improvement for Report Generation

For large projects with some thousand source files, HTML report generation including source code view is much faster now.
Bottleneck in this case was the navigation generated to browse between source files. You can achieve an additional performance improvement with an adapted HTML template omitting this navigation element.

Refinement of line execution information

In HTML reports, executable source code lines are highlighted in red (not executed), yellow (partially executed), or green (executed). With this version, more lines are recognized as partially executed. Line 6 in



is now categorized as partially executed. With earlier versions, this was not the case as the block `{x++;}` does not contain any counters. The block-closing "}" must be on the same line for this recognition to work.

Warnings of **ctc**

The distinction between "info" category and "warning" is harmonized: All currently existing non-error messages are "warnings" now. This can lead to more warnings with setting `WARNING_LEVEL = warn` in **ctc.ini**.
A new warning is introduced for C++ functions with specifier `constexpr`. These functions are not instrumented, and the new warning informs about their presence.

Changes in Windows Installer

- The complete package including HOTA libraries and BITCOV support is now always installed.
- When a local license file is used, the location for this file can be overwritten in dialogue.
- When installing over an existing installation, **ctcreport** folder, **ctclaunch.ini** and still **ctc.ini** are now copied on request. Contrary to previous versions, **ctc.ini** is now always renewed.

Removal of SOURCE_IDENTIFICATION

The configuration parameter SOURCE_IDENTIFICATION is no longer supported. During instrumentation, **ctc** always records full paths to source files in symbol file. This harmonizes many differences between included and directly instrumented files.

Removal of **ctcdiff**

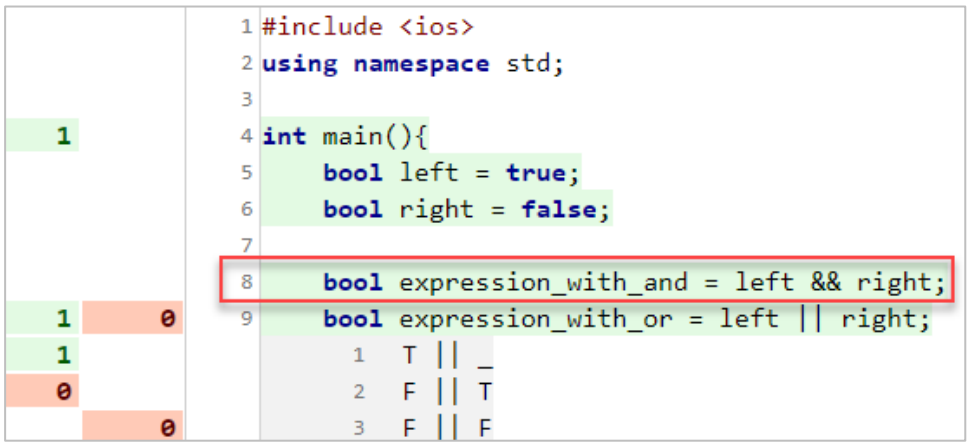
The tool **ctcdiff** for comparing **ctcpost**'s text reports of different test runs is no longer part of the Testwell CTC++ package. If you use it, please get in contact with us.

Bug Fixes

Not recognized Boolean expressions

Multicondition instrumentation was incomplete or missing, if the left-hand side of the and-operator corresponded to a type or a function from a namespace used with `using namespace`, or to a type in the same namespace.

Example:



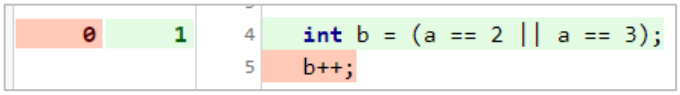
The expression in line 9 is correctly instrumented, but the one in line 8 was not. `left` was mixed up with `std::left` and the rvalue reference recognition introduced with version 9.1.1 caused a misinterpretation.

C++ spaceship operator

Spaceship operator `<=>` was processed as `<= >` by **ctc**, leading to uncompilable code.

Statement and line coverage after assignments

When a multicondition inside an assignment only evaluated to false, following statements were not counted as covered and corresponding lines were highlighted in red.



Parallel builds using BITCOV

For parallel builds with BITCOV instrumentation, it was possible that the generated auxiliary file **MON.aux** was corrupted. The locking mechanism in responsible script **ctc2static** is now improved.

Corrupted symbol file

During instrumentation, **ctc** did not record the symbol file correctly in a situation like

```
if (1)
   use_lambda([&] (void) -> void { ;});
else
   do_something();
```

A lambda function inside an `if` branch without `{…}` led to missing information for the `else` branch. In consequence, **ctcpost** miscalculated coverage for the corresponding function and **ctcreport** ended with an error.

Init-statements in decisions

With C++ 17 and beyond, init-statements may precede decisions in if-statements:

```
if (int x = foo(); a && b)...
if (foo(); a && b)...
```

For the second example, instrumentation led to uncompilable code.
When instrumented with multicondition, the first example was instrumented for decision coverage only.
Init-statements in range-based for loops (C++ 20) are now also supported, they led to a syntax error of **ctc**:

```
for (int the_answer=42; int r:my_array)...
```

Multiple source file representations in data files

In one data file (**MON.dat**), every source file is typically represented only once. However, the runtime-layer of Testwell CTC++ generates duplicates for source files present in an executable *and* in a DLL / shared object loaded by this executable. **ctcreport** could not handle this situation and ended with an error. The same error occurred when two source file identifiers in one data file were mapped to one target with option `-map-source-identification`.

Windows path spelling issues in HTML report

Differently spelled paths to source files during instrumentation, like

```
C:\projects\…
c:\projects\…
C:/projects/…
```

present in symbol and data files could lead to duplicated directories or files in HTML report.

Inactive lines from header files

Header files included more than once, but protected with include guards, could lead to source code lines wrongly assumed as inactive in HTML report. This behavior was compiler-dependent and for example present for Microsoft's Visual C++ compiler **cl**.

Handle-leaks of **ctcreport**

ctcreport could cause handle-leaks when processing symbol files, data files and report files.

Full support of code structures in symbol files by **ctcreport**

Following elements led to an error of **ctcreport** and are correctly processed now:

- function-try-blocks,
- Java `finally` statements,
- `#pragma CTC ANNOTATION …` located between `if`-branch and `else`.

Member initializing mixed up with function body

For certain member function definitions, **ctc** instrumented a member initialization as if it was a function body, leading to uncompilable code.
Example: `{h}` in

```
foo::Bar::Bar(Helper& h) : Base<Helper>{h} {…}
```

where `foo` is a namespace, `Base` a template class declared in this namespace and `Bar` a derived class.

Initialization of multidimensional arrays

Outside functions, initialization of multidimensional arrays with initializer lists like

```
int evil_array[3][5] {666};
```

could cause **ctc** to misinterpret the code as a lambda function.

Wrong function naming

Functions called like namespaces or classes got a wrong name by **ctc**. In this example

```
namespace one {
    /*...*/
}

namespace two {
    void one(){}
}
```

function `two::one()` was named as `one::one()`.

Declarations in `if` statements with BITCOV

When instrumented with BITCOV, counter information could get lost for false counters of if-decisions with a declaration inside like

```
if (int a = foo())
```

Additional variants

If any preprocessor variants of a source file were identified correctly by **ctcreport**, it was possible that additional variants were falsely identified. For the same set of source files, this could occur or not, purely dependent from build order.

Multiple function loops

In HTML templates for source code view, more than one function loop led to a **ctcreport** error.

Header extraction with **ctcpost**

The former way of header handling by **ctcpost** did not work properly for versions 10.0.0 and 10.0.1, due to changed symbol file format: Equal header copies were considered different and in consequence not extracted from their including source file.