22 January 2026

Verifysoft TECHNOLOGY

Change Documentation for

Testwell CTC++

Version 10.3.0

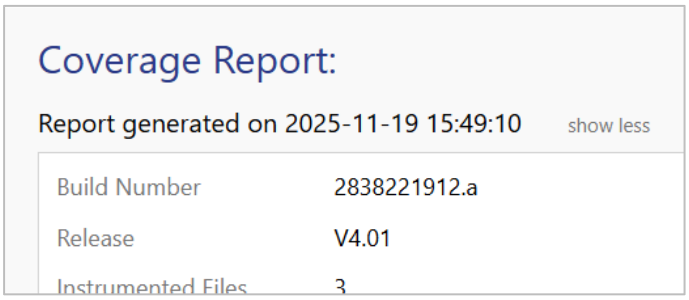
Features and Changes

User-defined variables in reports

When generating reports with **ctcreport**, additional information can now be handed over in key-value pairs with option `-D`:

```
ctcreport -D BuildNumber=2838221912.a -D Release=V4.01 …
```

In the HTML standard report, these variables are all displayed in the details block on the overview page:



Coverage Report:

Report generated on 2025-11-19 15:49:10 show less

Build Number	2838221912.a
Release	V4.01
Instrumented Files	3

In a reporting template, user variables can be addressed in two ways:

- Directly via their key, enclosed in $...$:
  `$BuildNumber$` is replaced with its value defined on command-line.
- All of them via a new loop:

  ```
  {CTC_LOOP("UserVariables")}
  $UserVariableLabel$: $UserVariableValue$
  {CTC_LOOP_END("UserVariables")}
  ```

  in a single-file text template leads to output

  ```
  Build Number: 2838221912.a
  Release: V4.01
  ```

  in the example above.

Three variables are available in the loop:

- $UserVariableKey$ is the key as defined with option -D.
- $UserVariableLabel$ is deducted from the key, adding blank spaces at each lower-to-upper case transition.
- $UserVariableValue$ is the current value of the variable.

Controlling source code paths during instrumentation

In addition to the mapping possibilities for source code paths during report generation, these paths can now be reduced to their stable part during *instrumentation*.
For this purpose, a new configuration parameter MASK_PATHS is introduced. It can contain a list of path anchors.

```
MASK_PATHS = MyProject, common\source
```

The full path of a source file is shortened when it contains one of these anchors. Everything left of the anchor is omitted in symbol file, object file, executable, and data file recording.

Hence, exactly reproducible builds in an environment with dynamically changing build directories are better supported.
A replacement is added to the left for internal processing, to distinguish Linux from Windows paths. When generating reports, use option `-map-source-identification` of **ctcreport** to assign the correct file location.

HTML / XML entities in single-file templates

When generating a report with a single-file template ending with `.xml` or `.html`, the following characters from function names, source code or user variable content are now replaced:

- < with &lt;
- > with &gt;
- & with &amp;

Remark: This behavior was already in place for structured templates.

Standard HTML report – additional information

In the overview page, the expandable box for detail information contains the following changes:

- All user variables are provided at the beginning of the table.
- Symbol and data files are now listed at the end, as there are often many of them.
- The version of **ctcreport** is reported at the bottom.
- The command-line cannot affect the box anymore, becoming very wide. A copy function and an expansion function are now provided; the latter wraps the command-line logically.

Change of caching behavior in ctc

During parsing of preprocessed source files, **ctc** caches paths to included files for performance reasons. This caching is now reset after each processed source file. This avoids a mismatch in drive letter capitalization on Windows.

Separation of Visual Studio integration

The Visual Studio integration is no longer delivered with the Windows version of Testwell CTC++. Instead, it can be downloaded separately from our customer area.

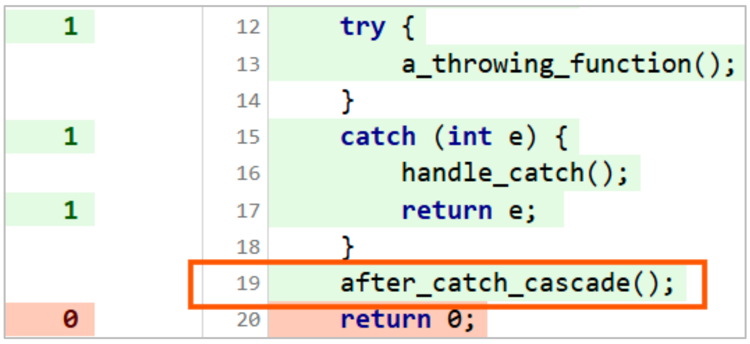
Withdrawal of ctcwrap

The tool **ctcwrap** is no longer part of the installation package. On Windows and Linux, use **ctclaunch** instead.
Get in contact with us early to discuss alternatives if you’ve been relying on **ctcwrap**.

Bug Fixes

Execution information after try - catch

After a try-catch cascade, code never executed could be falsely shown as executed. Line and statement coverage were affected:



```
1        try {
             a_throwing_function();
         }
1        catch (int e) {
             handle_catch();
1            return e;
         }
         after_catch_cascade();
0        return 0;
```

To derive this kind of coverage information, a new counter is introduced for the end of the try-block. After an upgrade, source code files containing try-catch will get a differing representation in symbol files, making them incompatible to coverage data collected in data files from older instrumentations. **ctcreport** of version 10.3.0 can handle both situations – hence you can keep existing symbol files (and prevent them from being overwritten) if you want to reuse test data.

Writing permissions for ctclog-folder on Linux

The folder `ctclog` in temp directory, dedicated for logs of **ctc** and **ctclaunch**, has now writing permission for all users. This avoids a crash of **ctclaunch** executed by another user after a first successful usage.

Missing Justification for merged MC/DC pairs

When merging was applied to justified counters, justifications could get lost for MC/DC criteria.

Lambda in Member Initializer

Since version 10.2.1, lambda functions inside member initializer lists are instrumented. Two issues were caused by this change:

- An instrumented program could crash if the lambda function was the first function called (of its compilation unit).
- Inside code *not* to be instrumented, **ctc** did not process the body of such lambda functions correctly. This could lead to subsequent faults. Observed were: Freezing of **ctc** during instrumentation and a misrecognition of a namespace end, the latter leading to function names prepended with `std::`.

Forwarding of build errors by ctclaunch on Linux

When the build system ended with an error code, **ctclaunch** on Linux did not always properly display and pass this error code.

Error of ctcreport

When called with `-merge-variants`, **ctcreport** could end with an error message:

```
Error retrieving data value with ID 189 (err=1)
```

This was caused by source files containing comments or other not preprocessed code at their end in combination with GNU compilers.

Wrong assignment of functions

With nested includes for instrumented header files, functions were sometimes assigned to the wrong source file.
In consequence, wrong variants were recognized by **ctcreport**. Only version 10.2.2 was affected by this issue.

Lambda functions with a templated return type

Lambda functions with a templated return type like

```
auto lambda =[]() -> MyType<template_param> {…
```

were not instrumented, if the type `MyType<template_param>` was not recognized properly by **ctc** (for any reason during typical header parsing).
This issue is fixed by a local improvement of lambda handling, independent of type recognition.