# Generating Coverage Information

This topic explains how to generate the coverage information from test sessions and how to adjust the setttings to minimize the execution overhead

## Enabling and Disabling Coverage Analysis

The instrumentation setting in the execution tab must be configured to collect code coverage information.  C++test provides an option for setting which coverage metrics should be collected during tested code execution. See Understanding Coverage Types for more information.

If performance is an issue, such as in embedded testing, you can also disable all code coverage monitoring. See Execution Tab Settings - Defining How Tests are Executed for more information about controlling coverage analysis.

Code coverage information can be collected from unit tests execution as well as from manual testing performed on your application built by C++test specifically for coverage monitoring.

## Understanding Code Coverage Analysis Overhead

By default, the code coverage information generated during tested code execution is immediately sent to C++test, which ensures accurate results. For example, if the test execution results in a crash, all coverage information up to the point where the crash occurred will be valid. The disadvantage of this approach is relatively high execution time overhead. The execution overhead for unit tests, however, is negligible in most cases. As a result, the default coverage mode is recommended.

Execution time overhead related to on-the-fly coverage data transfer (default mode), though, may not be acceptable for collecting coverage information from application level tests (application functional testing). For example, it is very common for coverage instrumentation to have an impact on application timing dependencies that affect the application logic in embedded systems.

If the default code coverage execution time overhead is unnaccetable, users can enable a special optimized mode to collect coverage from application level functional testing. In the optimized mode:

- Dedicated memory buffers are used to store information about covered C/C++ language constructions
- Coverage data is sent at application finalization;no t during functional tests execution
- Code instrumentation executes the minimum required amount of machine instructions to store the information about covered C/C++ language statements.
- Therefore there is no requirement to assure a fast link between host and target to minimize overhead.

## Optimized Coverage Metrics for Application Coverage Monitoring

The special optimized coverage instrumentation mode is only designed for application level testing and is not available for unit testing. Use the "Advanced options" from the Test Configuration's Execution Tab "Instrumentation features" panel to enable optimized coverage mode. These settings are described in Execution Tab Settings - Defining How Tests are Executed.

The "Advanced options" section enables you to:

- Choose lower tested code execution time overhead or lower overhead on ram memory.
- Initialize coverage memory buffers
- Protect against coverage data buffers corruption

Dedicated initialization and finalization functions must be called to run an optimized code coverage monitoring session:

```
void CppTest_InitializeRuntime(void)
void CppTest_FinalizeRuntime(void)
```

By default, these functions are called at the beginning and end of the main function (C language application) or via the special global object constructor and destructor added for this purpose (C++ language application).

If your application does not have a "main" function or does not support construction/destruction of global objects, you can manually inject calls into the initialization and finalization functions in appropriate locations during the application startup and finalization.