Running the Test and Reviewing Results

This topic explains how to create and run a test executable, then review results in the C++test GUI.

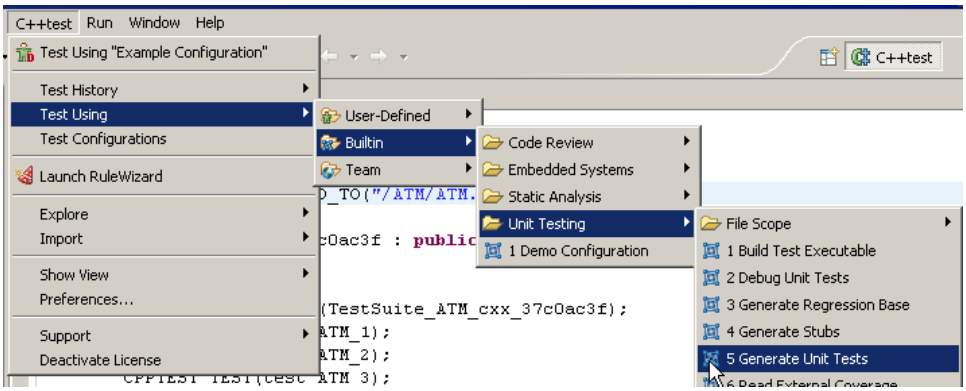
Sections include:

- Generating Tests
- Building the Test Executable
- Running the Test Executable and Reviewing Results
- Debugging Test Cases

Generating Tests

To generate test cases:

1. Run your preferred test generation Test Configuration (for example, choose **Parasoft> Test Using> Built-in> Unit Testing> Generate Unit Tests**).



Building the Test Executable

For most supported targets/platforms, the process of building the Test Executable is just a part of an appropriate "Run..."-like Test Flow.

There are few cases where it's convenient to have a separate "Build..."-like Test Configuration and the associated Test Flow. It's generally best to follow the suggested strategy for your platform. For platforms where the full "build, run, read results" flow is normally provided, use the separate build Test Configurations only in special cases—for example, when automatic executable-to-target loading isn't supported, or when you need only the static coverage data (data collected while building the test executable—without execution).

Such Test Configurations may be created by extracting the build-portions from the original Test Flows of the original Test Configuration or user-adjusted Test Configurations. In many cases, you need to adjust one of the provided builtin Test Configurations to make it work for you—creating a new user configuration. In other cases, you can just use the builtin original Test Configuration.

There are several ways to build the test executable:

- Choose **Builtin> Embedded Systems> (Platform)** and select a **Run** . . . Test Configuration (assuming that the subsequent run is acceptable).
- Choose **Builtin> Embedded Systems> (Platform)** and select a **Build** . . . Test Configuration.
- Choose **Builtin> Unit Testing> Build Test Executable** where it applies.
- Run a User-Defined Test Configuration.
- Run a user-defined Test Configuration with the build-portion of the flow cropped out of the original test flow.

Information on the Test Flow and its adjustment can be found in Customizing the Test Execution Flow**.**

Important Instrumentation Setting Considerations

Build process and subsequent execution are strongly affected by the instrumentation features accessible on the "Test Configurations> Execution" tab (see Fine-Tuning Test Settings for details. In embedded solutions, it's often convenient—or even required—to disable instrumentation of specific sources (for instance, containing startup code or interrupt-handling code). This may be achieved through the "Parasoft> C++test> Execution Settings..." context menu (right-click) action, which is available in the project tree; see Execution (File-Level Only) for details.

Running the Test Executable and Reviewing Results

If your test execution flow does not automatically deploy the test executable to the target device, start it, and/or read and display results in the GUI, you need to complete the necessary actions manually.

Tips for reading and displaying results via the file communication channel:

- The "Utilities> Load Test Results (File)" Test Configuration provides a default test execution flow to collect test results via the file channel. By default, this configuration assumes that logs are located inside ${cpptest:testware_loc}. If needed, you can customize this location to any file system location that can be accessed from the C++test GUI.
- Ensure that the log files can be accessed from the host. To achieve this, you can manually transfer the log files to a location that can be accessed from the C++test GUI. Or, you can customize the Test Configuration's test execution flow so this transfer is done automatically.

Tips for Reading and Displaying Results via the Socket Communication Channel

- The "Utilities> Load Test Results (Sockets)" Test Configuration provides a default test execution flow for "on the fly" collection of test results sent through TCP/IP sockets. It starts a java utility program to listen to and capture test results. You might need to customize the Test Configuration's test execution flow to use the correct host IP address and port values.
- The "Utilities> Load Test Results (Sockets)" Test Configuration must be started before the test executable.

"Cannot find test log file. ...failed" Message

If you receive this message, it may mean that 1) the test executable did not run correctly or 2) Test log files were not available in the expected location. In the latter case, the log files need to be transferred to the expected location OR the expected location needs to be changed to match the actual location.

Debugging Test Cases

Debugging in Various Embedded Development Environments

For some embedded environments, you can debug test cases directly in C++test. There are two modes for debugging test cases: 1) Internal mode and 2) External Embedded mode. For environments that C++test does not directly support test case debugging, you must exploit debugging methods for externally built executables provided by your environment (if available) and manually set breakpoints on wanted test cases. Check environment-specific chapters to see if C++test directly supports Test Cases debugging for your environment.

We strongly recommend using the built-in Test Configurations for particular embedded environments and making adjustments as necessary. See Executing Tests with a Debugger for instructions.

Enabling Internal Debugging Mode

1. Make a duplicate of the environment-specific Test Configuration and select the duplicate from the User-defined folder.
2. Choose the **Execution> Runtime** tabs and enable the **Run tests in debugger(*)** option.
3. Make sure that the **Use Eclipse internal debugger with configuration**: option is enabled.

See

Debugging in Eclipse Internal Debugger Mode

, for additional information.

External Embedded Debugging Mode

You can switch to External Native mode by modifying the test configuration. This mode is only used for debugging native applications. After switching to External Native mode, you must also modify the Test Flow recipe to enable External Embedded debugging mode:

1. Make a duplicate of the environment-specific Test Configuration and select the duplicate from the User-defined folder.
2. Choose the **Execution> Runtime** tabs and enable the **Run tests in debugger(*)** option.
3. Enable the **Use external debugger** option. *This enables External Native mode, which is only used for debugging native applications.*
4. Modify the Test Flow recipe to enable External Embedded. *This mode can only be selected by modifying the Test Flow recipe*. See Selecting External Embedded Debugging Mode, for instructions.