Executing Manually-Written CppUnit Test Cases

This topic explains how to prepare C++test to execute your existing manually-written CppUnit test cases. Running CppUnit test cases in C++test allows you to centralize unit testing and reporting. C++test's reporting and authorship calculation capabilities help the team track which test cases failed, since when, and who is responsible for fixing each failure.

The team can run these tests in command-line mode each night. For instant feedback on whether their code changes broke the existing functionality, each developer can import the regression failures caused by their modifications. Since the regression failures are directed to the developers responsible for them, the overall process of fixing them is much more streamlined than it would be if all developers were looking at the same list of regression failures.

Additionally, C++test provides test coverage information for CppUnit test cases and can also perform runtime error detection as they execute.

Sections include:

Accessing the Tests

To configure C++test to access your manually-written CppUnit test cases:

1. Ensure that the CppUnit test case files are available within the project tree.
    - The test directory can be located anywhere, as long as it is visible in the project tree. By default, C++test expects tests to be stored in a subdirectory of the project's tests directory. However, you can use a different location, as long as you modify the Test Configuration's **Tes t suite file search patterns** setting (in the **Execution> General** tab) accordingly.
    - If you do not want to store your tests within the project directory, you can add a folder that links to files stored elsewhere in your file system. To do this:

        a. Choose **File> New> Folder** (if this is not available, choose **File> New> Other**, select **General> Folder**, then click **Next**).
        b. Click the **Advanced** button.
        c. Enable the **Link to folderin file system** option.
        d. Enter or browse to the location of your source files.
        e. Click **Finish**.

2. Choose **Parasoft> Test Configurations** to open the Test Configurations dialog.

3. Select the Test Configurations category that represents the user-defined Test Configuration you want to execute the CppUnit tests.

4. Open the **Execution** tab.

5. In the **General** subtab, modify the **Test suite file search patterns** setting as needed so that C++test will locate and test the CppUnit tests.

    - Be sure to add an asterisk (*) at the end of the directory path.

6. Click either **Apply** or **Close** to commit the modified settings.

The tests will be executed when you run a test using this Test Configuration.

Requirements/Limitations

The imported test cases must satisfy the following conditions:

- The CppUnit test class must have CPPUNIT_NS::TestFixture or CPPUNIT_NS::TestCase as its base class.
- The CppUnit test class cannot be a template class.
- The CPPUNIT_TEST_SUITE_REGISTRATION(TestSuiteName) macro should be inserted into a CppUnit source file (.cpp).
- The following CppUnit macros are supported:
    - CPPUNIT_TEST_SUITE(ATestFixtureType)
    - CPPUNIT_TEST_SUITE_END()
    - CPPUNIT_TEST_SUITE_NAMED_REGISTRATION
    - CPPUNIT_TEST(testMethod)
    - CPPUNIT_TEST_EXCEPTION(testMethod, ExceptionType)
    - CPPUNIT_TEST_FAIL(testMethod) CPPUNIT_ASSERT(condition)
    - CPPUNIT_ASSERT_MESSAGE(message, condition)
    - CPPUNIT_FAIL(message)
    - CPPUNIT_ASSERT_EQUAL(expected, actual)
    - CPPUNIT_ASSERT_EQUAL_MESSAGE(message, expected, actual)
    - CPPUNIT_ASSERT_DOUBLES_EQUAL(expected, actual, delta)
    - CPPUNIT_ASSERT_THROW(expression, ExceptionType)
    - CPPUNIT_ASSERT_NO_THROW(expression)
    - CPPUNIT_TEST_SUITE_REGISTRATION(ATestFixtureType)
- The context of the imported CppUnit test cases will be set to 'project'.

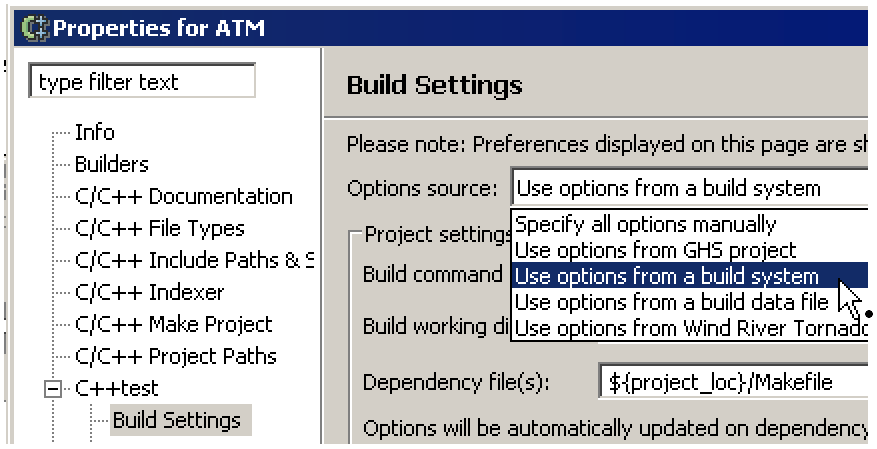
Excluding Test Cases from the Build

If a given CppUnit file is not excluded from build, then:

- Static analysis will be performed for that test file.
- Coverage will be reported from that test file.
- Function calls will be stubbed in that test file.
- Auto-generated tests will be created for that test file.

Thus, we recommend excluding CppUnit files from the build. However, it is not required.

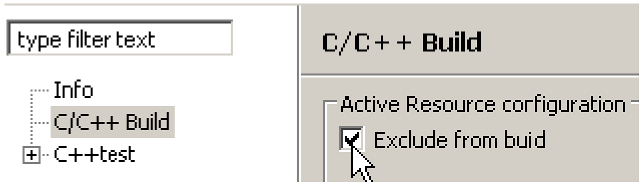
The procedure for excluding CppUnit files from the build depends on your project's options source (set in the C++test Build Settings):


Properties for ATM
type filter text
Info
Builders
C/C++ Documentation
C/C++ File Types
C/C++ Include Paths & S
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
C++test
Build Settings
Build Settings
Please note: Preferences displayed on this page are sh
Specify all options manually
Use options from GHS project
Use options from a build system
Use options from a build data file
Use options from Wind River Tornado
Project settings
Build command
Build working di
Options will be automatically updated on dependency

Options source: Use options from a build system

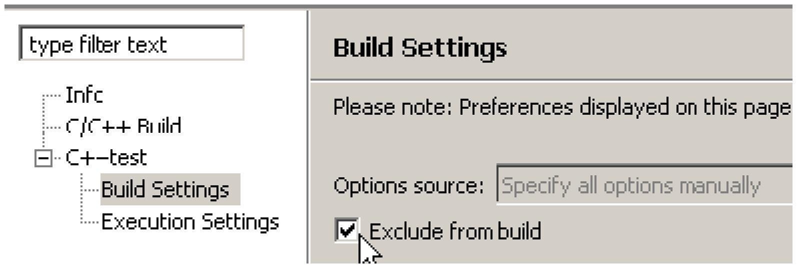
Dependency file(s): ${project_loc}/Makefile

- If the project is set to "Use Options from Managed Make C/C++ Project", right-click the source file containing the CppUnit method definitions, choose **Properties**, then go to **C/C++ Build> Exclude from build**. Then, repeat the same configuration for the header file containing the CppUnit class.



[✓] Exclude from buid

- If the project is set to "Specify All Options Manually", right-click the source file containing the CppUnit method definitions, choose **Properties**, then go to **Parasoft> C++test> Build Settings> Exclude from build**. Then, repeat the same configuration for the header file containing the CppUnit class.



Options source: Specify all options manually

[✓] Exclude from build

- For any other option source, the file will be excluded only if its build options are not available in the original build system (Makefile, .dsp etc.). For example, when you open the ATM project that is shipped with C++test, the CppUnit test suite is automatically excluded from build because it is not a part of the ATM project's Makefile.