Adding Test Suites and Test Cases with the Graphical Test Case Wizard

The Test Case Wizard allows you to select a function to test, then graphically specify test case preconditions and postconditions. These test cases can be parameterized with data source values (or automatically-generated values) to rapidly create test case scenarios that ensure broad, thorough test coverage, and that test the code's response to a wide range of inputs.

Test cases created in the Test Case Wizard are saved in source code, using the standard C++test test format (similar to CppUnit).

Adding Test Suites

All test cases must exist within a test suite. Before you can add test cases with the graphical Test Case Wizard, you must first have a test suite that can contain them.

Generating a Test Suite

To prompt C++test to generate a set of empty test suites for each testable context (file or function):

1. Select the resource(s) for which want to generate a test suite.
2. Run the "Unit Testing> Generate Test Suites" Test Configuration or a custom team Test Configuration that is based on it.
    - Any custom Test Configuration used for this purpose must be set to generate tests, but have the **Max. number of generated test cases (per function)** parameter set to zero.

Specifying a New Test Suite

If you want more control over the nature of the test suite than the automated generation (described above) allows, add a new test suite as follows:

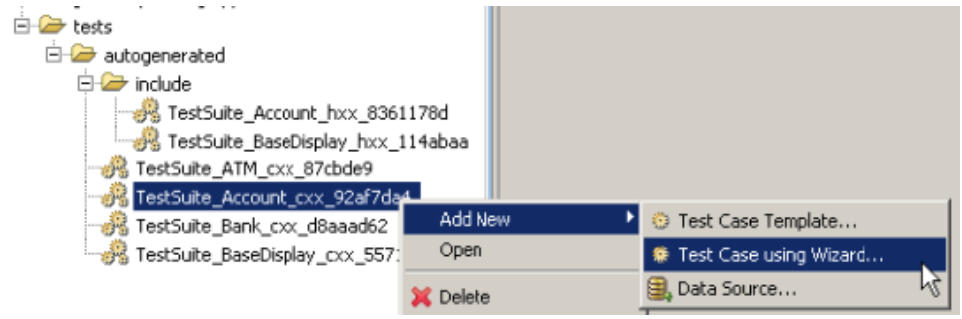
1. In the Test Case Explorer, right-click the node for the project that the test suite will test, then choose **Add New> Test Suite** from the shortcut menu
    - The Test Case Explorer is open by default, in the left side of the UI. If it is not available, you can open it by choosing **Parasoft> Show View> Test Case Explorer**. For details on understanding and navigating the Test Case Explorer, see the Exploring the C++test UI.
2. Enter the test suite parameters in the Test Suite Wizard dialog. Available parameters are:
    - **Test suite name**: Determines the test suite's name.
    - **Test suite location**: Determines the test suite's location.
    - **Test suite file(s)**: Reports the path to the test suite file(s).
    - **Test suite language**: Determines whether the test suite is implemented in C or C++.
    - **Tested file**: Sets the test suite context to the specified tested source file. The specified file will be set as the CPPTEST_CONTEXT macro, which associates a given test suite file with a specified source file.
        - If a file was selected when you started the wizard, that file will be specified here.
        - If a project was selected when you started the wizard, no context will be spec-ified, and the test suite will be at the project scope (which means it will only get executed if all tests for the project are run, or if it is selected as the test file). When it is selected as the test file (and does not have the CONTEXT) C++test assumes it has the project context, and prepares all files in the project to link against.
    - **Test suite mode**: Determines whether the test suite is implemented as an included test suite, or a standalone test suite. C++test instruments both types of test suites; it can access private/protected class members.
        - Included test suites are physically included in one of the generated test har-ness source files. "Included" test suites are combined with instrumented code and compiled to form a single object file. All automatically-generated test suites are included test suites. Additional header files can be included and macro definitions can be defined in original headers. However, included test suites do not need any headers included, unless there are types used that are not visible in the original tested source file. Typically, this is only necessary when you are modifying generated tests (e.g., to include a test factory, etc.).
        - Standalone test suites are compiled separately and linked in with the test exe-cutable for the testable context. Any additional headers must be included directly in these test suites. Coverage information can be tracked for included headers.
    - **When referencing tested file in test suite**: Determines whether the CPPTEST_CONTEXT and CPPTEST_TEST_SUITE_INCLUDED_TO macros use the full project path or relative paths.
        - In most cases, using full paths is recommended. Relative paths can be helpful in special cases, such as when you want to generate tests for code that is used in different locations, and you want to use the tests in multiple locations as well. For example, assume that you have source files for a library that can be used by many different projects, and you have some tests connected with that source code. In that case, no matter where that source code is placed in the projects, the connected tests should work with it (because no full paths are used).

Adding Test Cases Using the Graphical Wizard

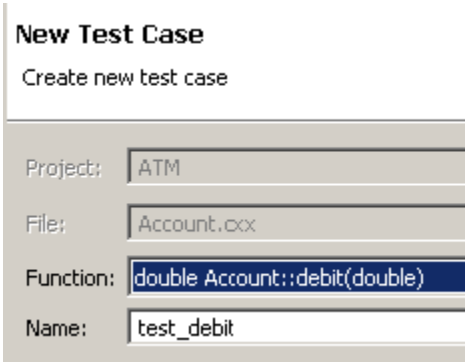
You can use the Test Case Wizard to specify new test cases graphically, using GUI controls. C++test generates test case code representing the specified tests, and adds this code to the corresponding test suite. These test cases can be executed along with the other test cases, and they can be modified /extended as needed.

1. To add a new test case using the Test Case Wizard:
    - The Test Case Explorer is open by default, in the left side of the UI. If it is not available, you can open it by choosing **Parasoft> Show View> Test Case Explorer**. For details on understanding and navigating the Test Case Explorer, see the About the Parasoft Test UI.

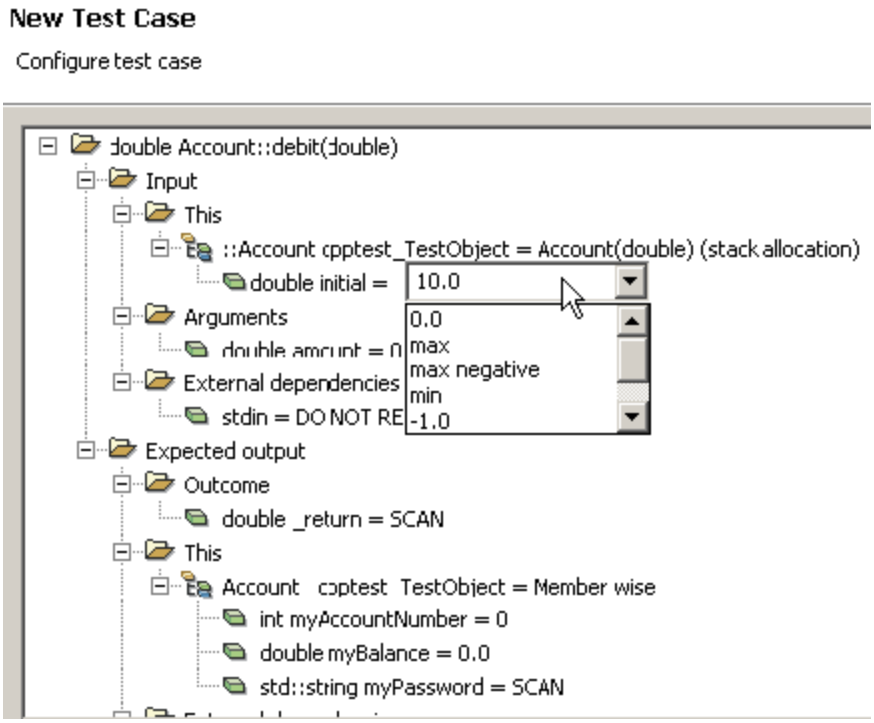
2. Right-click the selection, then choose **Add New> Test Case using Wizard** from the shortcut menu.



3. On the first page, specify the source file (compilation unit) and the function for which you want to add a test case, then enter a name for the test case.
    - The option to specify a source file is available only if your test suite was configured as a "standalone" test suite.
    - If the test suite was configured as "included", you must choose among functions from the compilation unit that the Test Suite is included to.
    - If the test suite has a context file specified, you must choose among functions defined in the given source/header file.



4. Click **Next** to open the next wizard page.
5. Configure the test case by specifying its input and expected output values using GUI controls.
    - See Test Case Configuration Tips for details.



6. Click **Finish** to generate the test case. The new test case will be added to the test suite and the generated source code will be opened in the editor.

Test Case Configuration Tips

- You can specify the following pre-conditions for the test case:
  - arguments for the tested function
  - (for the non-static member functions) value of the tested object ('this')
  - values of global variables used by the tested function
  - value of the standard input stream (see Using Data From Standard IO)
- In addition, you can specify the following expected post-conditions:
  - return value of the tested function
  - (for non-const reference and pointer types) values of the tested function arguments
  - (for the non-static member functions) state of the tested object ('this')
  - values of global variables used by the tested function
  - value of the standard output / standard error streams (see Using Data From Standard IO)
- To specify a description for a test case, add it in the **Additional settings> Test case description** field. It will be saved with the generated test case.
- To have C++test insert macros that report the values of test case inputs and outputs, enable the **Additional settings > Insert code to report test case inputs and outputs** option.
- To change the value for a given pre- or post-condition, double-click it then select the appropriate initializer type.
- For simple type nodes (boolean, integer, floating point, string types), you can provide an appropriate value by editing the node value.
- Values that do not need to be initialized (e.g., global variables or the value of the standard input stream) can be left uninitialized by selecting the 'LEAVE NOT INITIALIZED' or 'ACTUAL' value.
- To exempt a post-condition node from verification, select the 'ANY' value. This will prevent C++test from generating any assertion macros for the given outcome.
- For each simple type post-condition node (boolean, integer, floating point, string types) as well as for post-conditions of pointers to simple types (e. g. char* or int*), you can have C++test scan the actual value by selecting the 'ACTUAL' value. This will prompt C++test to generate a post-condition macro instead of the assertion for the given outcome in the resulting test case.
- To learn how to use data sources values in test cases, see Using Data Source Values to Parameterize Test Cases.
- If C++test is unable to provide any valid initializer for a given test case object (e.g. to create a reference to a forward-declared class), then the related test case tree node will be marked with a red X icon. This icon will also mark all parent nodes up to the top-level node representing the tested function. Generating a test case in this state will result in C++test creating an incom-plete test case that will need to be modified manually.