Generation Tab Settings - Defining How Test Cases are Generated

During a test, C++test will create test cases based on the criteria defined in the selected Test Configuration's Generation tab.

The Generation tab has the following settings:

- **Enable Unit Test Generation:** Determines whether C++test generates test cases for code in the test scope. If this option is not checked, all other test generation parameters are irrelevant.

General tab

- **Generate tests for code:** Determines when tests are generated. See Common Test Generation Goals for help on choosing the appropriate settings. Available options include:
    - **Without test suites:** Prompts C++test to generate tests for code that does not have any valid test suites. C++test searches for available test suites based on the criteria specified in the **Execution> General** tab.
    - **With out-of-date test suites:** Prompts C++test to generate tests for code that has been modified since the time that the related tests suite was generated. Old tests will be overwritten.
    - **With up-to-date test suites:** Prompts C++test to generate tests for code that has not been modified since the time that the related test suite was generated.
- **Generate tests for function access level:** Determines whether C++test generates tests for public/global functions, protected functions, and private functions.
    - The **Access to private members** instrumentation option must be enabled in order to generate tests for protected functions or private functions. See Execution tab for details on this option, which is controlled by **Execution> General> Instrumentationmod**
      .
- **Max. number of generated test cases (per function):** Determines the maximum number of test cases that C++test will generate per function.
- **Add test case description:** Determines if generated test cases will have the description that you provided.

Test suite tab

- **Test suite output file and layout**: Determines the test suite's file name, location, and granularity/layout. See Customizing Generation Options for details.
- **When generating tests for code with an existing test suite:** Determines how C++test handles new test cases that are generated for code with an existing test suite. The new test cases can either replace the existing test suite, or be integrated into it. See Common Test Generation Goals for help on choosing the appropriate settings. Available options include:
    - **Add tests for functions without tests:** C++test will generate test cases for functions without tests. The existing tests will not be affected or modified.
    - **Add tests for all functions:** C++test will generate test cases for all functions. The existing tests will not be affected or modified.
    - **Replace the existing test suite:** C++test will generate test cases for all functions. The existing test suite will be removed and then replaced with the new one.
    - **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).

Test case tab

- **Initialize global variables as test preconditions:** Determines whether global variables are initialized. When this option is enabled, automatically-generated test cases will initialize related global variables as preconditions.
- **Use objects of derived classes:** Determines whether objects of derived classes defined in the tested compilation unit should be used as input values. When this option is enabled, C++test will also use objects of such derived classes as primary test objects when testing methods of abstract classes.
- **Use heuristics for input values:** Determines whether heuristics values should be used for input values.
- **Use non-public class members in pre/post-conditions:** Determines whether non-public class members are used in pre /postconditions. For instance, it determines if C++test can use private constructors to create an object of a class type, use private fields in memberwise precondition initialization, or display values of private fields as outcomes in test case postconditions.
    - The **Access to private members** instrumentation option must be enabled in order to use non-public class members in pre /postconditions. See Execution tab for details on this option, which is controlled by **Execution> General> Instrumentation mode**.

- **Use null values for pointers:** Determines whether C++test uses null values to initialize pointer objects in test case preconditions. If it is disabled, null pointers will not be used in automatically-generated test cases.
- **Use member-wise initialization for class/struct objects:** Determines whether class/struct objects are initialized in preconditions by varying non-static member fields. In this case, an object is created by using the default constructor and assigning a value to each of the class/struct member fields.
- **Use factory functions:** Configures C++test to use factory functions, which are described in Using Factory Functions. If you want to use factory functions exclusively, also enable **Do not use other initializers for types with factory functions**.
- **Redirect stdin/stdout/stderr streams:** Determines if C++test automatically inserts streams redirection code to auto-generated test cases (you can add some data to stdin and/or check stdout/err as post-conditions). The test case code will look something like:

```
...
   CppTest_StreamRedirect* _stdinStreamRedirect =
CppTest_RedirectStdInput("some value");
   CppTest_StreamRedirect* _stdoutStreamRedirect =
CppTest_RedirectStdOutput();
   CppTest_StreamRedirect* _stderrStreamRedirect =
CppTest_RedirectStdError();
   ...
   /* Tested function call */
   ...
   /* Post-condition check */
   ...
   CPPTEST_POST_CONDITION_CSTR_N("stdout", (CppTest_StreamReadData
(_stdoutStreamRedirect, 0) ), 256)
   CPPTEST_POST_CONDITION_CSTR_N("stderr", (CppTest_StreamReadData
(_stderrStreamRedirect, 0) ), 256)
```

- **Memory allocation for test objects:** Determines whether objects are created on the stack or on the heap.
- **Buffer allocation size:** Determines the number of elements that should be allocated when creating buffers of simple types (such as char or int) in preconditions.
- **Insert code to report test case inputs and outputs:** Determines whether macros reporting values of test case inputs and outputs are inserted into automatically-generated test cases.
- **Insert code to report test case outcomes:** Determines whether macros reporting test case outcomes (post-conditions) are inserted into automatically-generated test cases. Such macros can be later verified and automatically converted into assertions.
- **Test case outcomes display settings:** Determines the maximum number of characters to display for string types and the display method for simple type pointers. For the latter, it allows you to specify whether C++test should display first element or a fixed number of bytes for the simple type pointer allocation.
- **Insert assertion templates:** Determines whether (commented out) assertions are inserted into automatically-generated test cases.