Using Stub Callbacks

This topic explains how to use Stub Callbacks for dynamic stubs configuration.

In this section:

Introduction

Stub Callbacks provide a powerful mechanism for programming test case-specific stub logic. For each test case, you can define specific stub behavior to be performed each time the stub is called during test case execution. C++test automatically generates code that runs the test case-specific stub logic, so you do not need to modify the main stub definition.

You can configure Stub Callbacks in one of the following ways:

- Using the interface of the Test Case Editor (see Configuring Stub Behavior in the Test Case Editor)
- Manually modifying test case source code (see Configuring Stub Behavior in Source Code)

The Test Case Editor automatically generates configurable components of the Stub Callback framework, displays suggestions for stub configuration, and registers Stub Callbacks once they are created.

What if test case-specific stub logic is not defined?

Each C/C++test's stub has its default logic built into the stub definition – typically, a stub only returns simple/default return values. If a stub does not have a Stub Callback Function defined, C++test will use the default logic when the stub is called. In addition, you can configure C/C++test to call the original function if test-specific stub logic is not defined; see Creating Stubs that Call the Original Function.

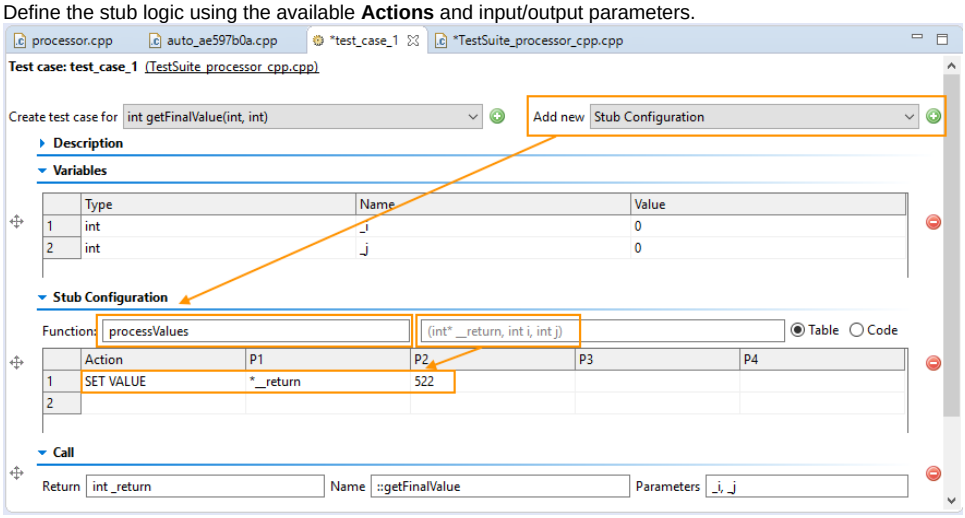
Quick Start with Stub Callbacks

Before you create a stub, ensure that the **Enable Stub Callbacks** option is enabled in the Stubs view (see Configuring Stub Options in the Stubs View) or in your test configuration (see Execution Tab Settings - Defining How Tests are Executed).

Configuring Stub Behavior in the Test Case Editor

1. Create a stub for the function you want to configure with the **Generate Auto Stub** or **Create User Stub** option (see Adding and Modifying Stubs).
2. Open test case in the Test Case Editor.
3. Add a new **Stub Configuration** step and specify the **Function** you want to configure.

4. Define the stub logic using the available **Actions** and input/output parameters.



5. Execute your test cases. When the test case is run, the custom stub logic defined in your Stub Configuration step will be executed when the stub is called.

See Adding Test Suites and Test Cases with the Test Case Editor - Configuring Stub Behavior for more details about configuring stub behavior using the Test Case Editor.

Configuring Stub Behavior in Source Code

1. Create a stub for the function you want to configure with the **Generate Auto Stub** or **Create User Stub** option (see Adding and Modifying Stubs).

```
/** Auto-generated stub definition for function: int processValues(int, int) */
int processValues (int i, int j) ;
int CppTest_Auto_Stub_processValues (int i, int j)
{
    CPPTEST_STUB_CALLED("processValues");

    int __return = 0;

    /**
     * This section enables Dynamic Stub Configuration with Stub Callbacks.
     *
     * IMPORTANT: THIS COMMENT BLOCK SHOULD NOT BE DELETED OR MODIFIED
     *
     * 1. Define stub callback function in test suite file - use the following signature:
     *     void CppTest_StubCallback_SomeName(CppTest_StubCallInfo* stubCallInfo, int* __return, int i, int j)
     *
     * 2. Register stub callback in test case function - use the following code:
     *     CPPTEST_REGISTER_STUB_CALLBACK("processValues", &CppTest_StubCallback_SomeName);
     *
     * 3. Fill out the body of the stub callback function according to intent.
     * The following line may be used to call original function inside stub callback:
     *     *__return = ::processValues(i, j);
     */
    if (CPPTEST_STUB_HAS_CALLBACK()) {
        CPPTEST_STUB_CALLBACK_PARAMS(int* __return, int i, int j);
        CPPTEST_STUB_INVOKE_CALLBACK(&__return, i, j);
    }

    else {
        /* You can put additional stub logic here. */
    }

    return __return;
}
```

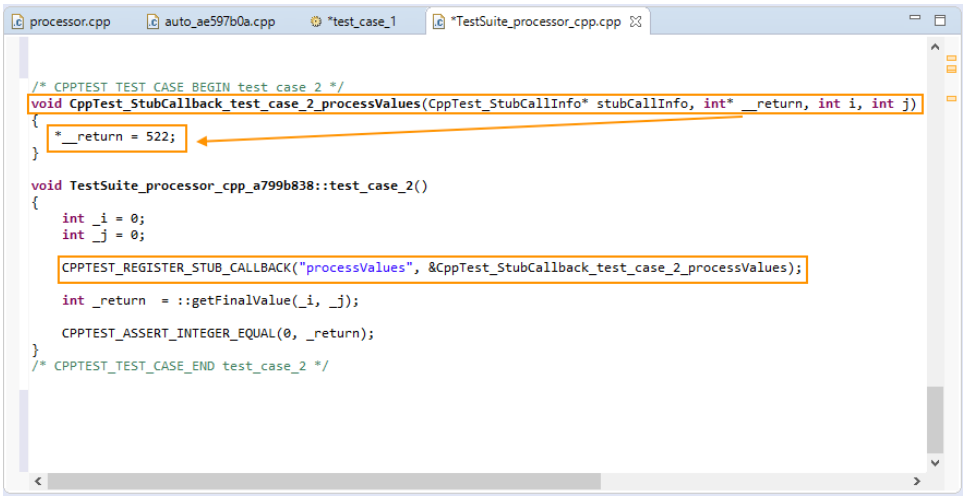
2. Open test suite file in the Code Editor.
3. Add a Stub Callback function to the test suite file.
4. Define the stub logic in the Stub Callback function, using the available input/output parameters.

> ✅ You can copy the Stub Callback signature from the stub file. See Stub Callback Execution.
>
> ✅ You can modify the name of the Stub Callback function, if needed.

5. Navigate to the test case definition and register the Stub Callback using the following macro:
`CPPTEST_REGISTER_STUB_CALLBACK(<STUB_ID>, &<STUB_CALLBACK_NAME>)`

> ✅ You can copy the stub identifier `<STUB_ID>` from the stub file. See Stub Callback Execution.

Ensure that the Callback is registered before the function under test is called.



6. Execute your test case. When the test case is executed, the stub logic you configured will be executed when the stub is called.

See Stub Callback Details for more information about Stub Callback Function and Stub Callback Registration.

Stub Callback Details

The Stub Callbacks framework consists of the following components:

- Stub Callback Function: a user-defined function that implements the custom stub logic for individual stubs.
- Stub Callback Registration: a mechanism that activates Stub Callback Function, which is typically placed inside a test case.
- Stub Callback Execution: an invocation of Stub Callback Function, which is built into the C++test's stub definition.

Stub Callback Function

The Stub Callback Function is a user-defined C/C++ function that contains logic to be executed when the related stub is called. It has a unique signature derived from the signature of the stub/stubbed function:

```
void <CALLBACK_NAME>(CppTest_StubCallInfo* stubCallInfo, <STUB_IN_OUT_PARAMETERS>)
```

The interface of the Stub Callback Function includes all the input/output parameters of the stubbed function (`<STUB_IN_OUT_PARAMETERS>`), as well as additional helper parameters (`stubCallInfo`). You can also customize the name of the Stub Callback Function (`<CALLBACK_NAME>`).

The following example demonstrates how the Stub Callback Function may be customized for a `int processValues(int i, int j);` function:

```
void CppTest_StubCallback_processValues_case_1(CppTest_StubCallInfo* stubCallInfo, int* __return, int i, int j)
{
    // checking input value
    CPPTEST_ASSERT_INTEGER_EQUAL(-1, i);
    // modifying return value
    *__return = 150;
}
```

We recommend that you keep the Stub Callback Function together with the related test case definition in the same test suite file.

You can define the Stub Callback Function as a global function or as a static member function (C++ only) of the test suite class. This allows the Stub Callback Function to access private class members of the code under test when the "Access to private members code" instrumentation feature is enabled.

The custom stub logic can be expressed with any valid C or C++ code using the following Stub Callback interface:

- stubbed function parameters – available as parameters of the Stub Callback Function
- stubbed function return value – available as the "`__return`" pointer to the return value holder
- stubbed function class object – available as the "`__this`" pointer to the class object
- the current call number of the Stub Callback Function – available as the "`int stubCallInfo->callNo`" variable

In addition, you can use a global variable or global function in the Stub Callback Function, including the C++test Unit Testing API (e.g., `CPPTEST_ASSERT` and `CPPTEST_REPORT`). If the original definition of the stubbed function is available, it can also be executed from the Stub Callback Function.

You can define multiple Stub Callback Functions in a Test Suite file — they can correspond to the same stub or a number of different stubs.

One Stub Callback Function can be used in a single test case or shared across a number of different test cases.

A stub can have only one Stub Callback Function active at a time (see Stub Callback Registration).

Stub Callback Registration

For a Stub Callback Function to be active, it must be registered. The function is typically registered inside a related test case before the first call to the stubbed function. In most scenarios, registration should be placed before the call to the function under test.

Use the following API call to register a Stub Callback Function:

```
CPPTEST_REGISTER_STUB_CALLBACK(<STUB_ID>, &<STUB_CALLBACK_NAME>)
```

where:

- `<STUB_ID>` is the unique stub identifier (see Stub Callback Execution)
- `<STUB_CALLBACK_NAME>` is the name of the user-defined Stub Callback Function

For example:

```
CPPTEST_REGISTER_STUB_CALLBACK("processValues", &CppTest_StubCallback_processValues_case_1);
```

You can register the same Stub Callback Function in mulitple test cases so that each of them will use the same custom logic for a given stub. Also, it is possible to register different Stub Callback Functions (for different stubs) in a single test case. This enables the test case to provide custom behavior for multiple stubs. At any given time, there can be only one Stub Callback Function active for given stub.

The lifetime of a Stub Callback Function:

- activation: registration with `CPPTEST_REGISTER_STUB_CALLBACK(<STUB_ID>, &<STUB_CALLBACK_NAME>)`
- deactivation: end of a test case or registration of another Stub Callback Function for the same stub (same `<STUB_ID>`)

Example:

```
void test_case_1()
{
    ...
    // CppTest_StubCallback_foo activation:
    CPPTEST_REGISTER_STUB_CALLBACK("foo", &CppTest_StubCallback_foo);
    // CppTest_StubCallback_bar activation:
    CPPTEST_REGISTER_STUB_CALLBACK("bar", &CppTest_StubCallback_bar);
    ...
    callToFunctionUnderTest(); // will call foo() and bar()
    ...
} // end of test case: CppTest_StubCallback_foo deactivation, CppTest_StubCallback_bar deactivation


void test_case_2()
{
    ...
    // CppTest_StubCallback_foo activation:
    CPPTEST_REGISTER_STUB_CALLBACK("foo", &CppTest_StubCallback_foo);
    // CppTest_StubCallback_bar activation:
    CPPTEST_REGISTER_STUB_CALLBACK("bar", &CppTest_StubCallback_bar);
    ...
    callToFunctionUnderTest(); // will call foo() and bar()
    ...
    // CppTest_StubCallback_foo_1 activation, CppTest_StubCallback_foo deactivation
    CPPTEST_REGISTER_STUB_CALLBACK("foo", &CppTest_StubCallback_foo_1);


} // end of test case: CppTest_StubCallback_foo_1 deactivation, CppTest_StubCallback_bar deactivation
```

Stub Callback Execution

If the Stub Callback Function is defined and registered, it will be executed each time corresponding stub is called. The Callback Function is invoked by calling `CPPTEST_STUB_INVOKE_CALLBACK()`, which is automatically added into a stub definition if the "Enable Stub Callbacks" option is enabled when the stub is created (this option is enabled by default).

Example:

```
/** Auto-generated stub definition for function: int processValues(int, int) */
int processValues (int i, int j) ;
int CppTest_Auto_Stub_processValues (int i, int j)
{
    CPPTEST_STUB_CALLED("processValues");

    int __return = 0;

    /**
     * This section enables Dynamic Stub Configuration with Stub Callbacks.
     *
     * IMPORTANT: THIS COMMENT BLOCK SHOULD NOT BE DELETED OR MODIFIED
     *
     * 1. Define stub callback function in test suite file - use the following signature:
     *     void CppTest_StubCallback_SomeName(CppTest_StubCallInfo* stubCallInfo, int* __return, int i, int j)
     *
     * 2. Register stub callback in test case function - use the following code:
     *     CPPTEST_REGISTER_STUB_CALLBACK("processValues", &CppTest_StubCallback_SomeName);
     *
     * 3. Fill out the body of the stub callback function according to intent.
     * The following line may be used to call original function inside stub callback:
     *     *__return = ::processValues(i, j);
     */
    if (CPPTEST_STUB_HAS_CALLBACK()) {
        CPPTEST_STUB_CALLBACK_PARAMS(int* __return, int i, int j);
        CPPTEST_STUB_INVOKE_CALLBACK(&__return, i, j);

    } else {
        /* You can put additional stub logic here. */

    }

    return __return;
}
```

The Stub Callback invocation section that is automatically added to stub definition contains a default signature of the Stub Callback Function and the exemplary Stub Callback Registration line that can be copied into test suite file when you configure Stub Callbacks manually.

Stub Identifier

Stub identifiers that are used for Stub Callback Registration and Stub Callback Execution must be unique to identify individual stubs. By default, stubs generated by C++test use identifiers according to the following pattern:

```
[<parent::>]<function_name>
```

The `<parent::>` prefix is added only if the stubbed function is a class or namespace member. Only the direct parent is added. In the case of template class methods, template parameters are omitted in the parent name.

Using Stub Callbacks for Overloaded Functions

C/C++test uses the same stub identifier for all overloaded versions of the function (see Stub Identifier). Before using Stub Callbacks for overloaded functions, update identifiers used for Stub Callback Execution code (`CPPTEST_STUB_CALLED()`) and Stub Callback Registration to ensure they are unique. This will allow you to distinguish stubs for overloaded functions/methods.

Example:

The default identifier used for all overloaded versions of foo():

```
CPPTEST_STUB_CALLED("foo");
```

The updated identifier for foo(int):

```
CPPTEST_STUB_CALLED("foo_int")
```

Creating Stubs that Call the Original Function

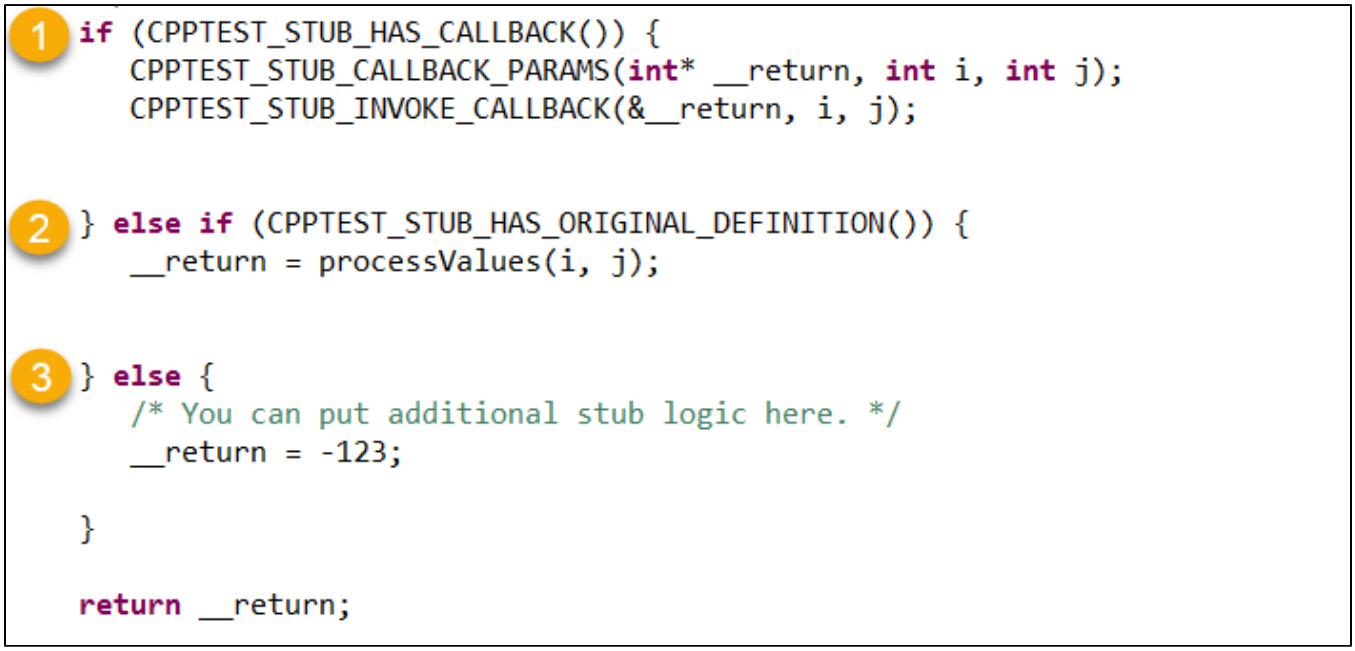
C/C++test allows you to generate stubs that call the original function if no test-case specific Stub Callback Function is registered in the test case. To achieve this, enable following options in the Stubs view (see Configuring Stub Options in the Stubs View) or in your test configuration (see Execution Tab Settings - Defining How Tests are Executed):

- **Enable Stub Callbacks**
- **Insert call to original function**

With these options enabled, C/C++test will apply the following stubbing strategy:

1. If a test case-specific Stub Callback is registered for the currently executed test, the Stub Callback Function will be used to stub the original function.
2. If no test case-specific Stub Callback is registered for the currently executed test, the stub will call the original function. As a result, the original function will be called during the test case execution, with the stub acting as a "proxy".
3. If no test case-specific Stub Callback is registered for the currently executed test and the the original function is not available, C/C++test will use the custom logic you can specify in the stub definition.

Example:



Calling original functions is not supported for the following:

- constructors
- functions with ellipsis
- pure virtual functions
- functions with internal linkage
- functions that have a deduced return type and are not visible from any header