Tutorial

Return Value Sequence for Stubs

If a test object (i.e. the function under test) itself calls other functions, Tessy can provide stub functions for the functions called. This tutorial shows how to provide different return values for a stub function, when it is called several times during the execution of a test case.

Furthermore, the Call Trace feature of Tessy is mentioned.

Architecture: TESSY
Author: Frank Buechner
Revision: 12/2013 – 005
1 Return Value Sequence for Stubs

1.1 The Problem

The test object calls another function several times. The function has to return different values during the execution of a test case.

```c
// For an reasonable test of dispatch1(), the stub
// of get_job() has to return different values for
// a single test case.
// (c) Hitex Development Tools GmbH, 2004
extern void func1(void);
extern void func2(void);
extern int get_job(void); // returns 0 if nothing left to do

int dispatch1(void)
{
    int job;
    while ((job = get_job()))
    {
        switch(job)
        {
            case 1: func1(); break;
            case 2: func2(); break;
            default: return job;
        }
    }
    return job;
}
```

Fig. 1 Example: The function dispatch1() calls get_job()

The test object dispatch1() calls the external function get_job(). If get_job() returns 1, func1() will be called. If get_job() returns 2, func2() will be called. If get_job() returns 0, the while-loop will be exited.

Semantics: The function get_job() shall return a sequence of 1 and 2, as long there is some work to do by func1() resp. func2(). If there is nothing left to do, get_job() shall return 0. This causes the while-loop to finalize and dispatch1() to return 0 (by the return statement at the end of dispatch1()), what is considered to be the normal termination of dispatch1(). If get_job() returns an illegal value (i.e. any value besides 0, 1, 2), the default case of the switch statement is executed, and the illegal value is returned by dispatch1(), what in turn shall indicate an abnormal termination of dispatch1().

For an reasonable test of dispatch1(), get_job() has to return different values during the execution of a single test case. If get_job() would return a fixed value, either (a) neither func1() nor func2() would be called, or (b) always func1() would be called, or (c) always func2() would be called. Both (b) and (c) would result in an infinite while-loop and the test would never finish.
1.2 Objective

The objective of this tutorial is to show how to specify a sequence of different return values for get_job().

Please note:
The feature described in this tutorial was not available in Tessy V2.9. With Tessy V2.9 you need to use a solution, in which the body of a stub function has to be programmed by the user. This solution is described in a former application note from Hitex.

2 Solution

Tessy can handle calls of the test object to other functions by providing stub functions. There are two types of stub functions that Tessy can provide:

- stubs with user code and
- stubs with parameter check and return value specification

These two types are depicted in the left hand part of the figure below.

![Handling called functions](image)

Fig. 2 How to handle functions called by the test object

For stubs with user code (also known as “normal” stub functions or “user stubs”) the user may provide the C source code for the body of the stub.

For stubs with parameter check and return value specification (also known as “advanced” stub functions) the user does not need to specify source code. Tessy automatically creates the code that is necessary to check the value of parameters against expected values and to provide the specified return values.

If Tessy is not directed by the user to create stub functions, the user has to take care to provide the called functions (right hand part of the figure above). This is not handled in the tutorial at hand.
2.1 Create Stub Functions

Fig. 3 The initial interface of dispatch1()

The test object dispatch1() calls three other functions: func1(), func2() and get_job(). These three functions normally will be implemented in some other part of the application under test. However, for the purpose of unit testing, we do not want to use these implementations. Therefore, we create stub functions as a replacement for the three called functions.

We create user stub functions (i.e. stubs for which the user could provide source code for the body of the stub) for func1() and func2().
2.1.1 User Stubs for func1() and func2()

![Interface of 'dispatch1']

Fig. 4 We create a user stub function for func1()

If Tessy is directed to create a user stub function, the Tessy user is able to provide C source code for the body of the stub function. This can be done in the Usercode view of the TDE perspective. Tessy generates the signature of the stub function and the curly braces for the test application. Tessy does not generate the body of the stub.

```c
void func1(void)
{
}
```

Fig. 5 The user stub function for func1() as created by Tessy

**Please note:**
Because we currently do have only one test object it makes no difference if we use “Create Stub (for all test objects)” or if we use “Create Stub (for current test object)”.
Fig. 6   We create a user stub function for func2().

Please note:
The icon for func1() has changed. The current icon indicates that Tessy will create a user stub for func1().

Fig. 7   The Stub Functions view in the TDE perspective of Tessy

It would be possible for the user to enter C code in the field “Sub Code”. For the tutorial at hand this is not necessary, because (a) func1() is a void function and (b) we do not want to do func1() anything.
2.1.2 Advanced Stub for `get_job()`

Advanced stub functions allow to specify return values (returned from the advanced stub function to the test object) and to check parameter values (passed from the test object to the stub function). The passing direction is as seen from the test object, i.e. parameter values have the passing direction “OUT” and return values have the passing direction “IN”.

![Diagram](image)

Fig. 8 Passing directions of variables of advanced stub functions
Fig. 9  We create an advanced stub for get_job()

Please note:
Because we currently do have only one test object it makes no difference if we use “Create Advanced Stub (for all Testobjects)” or if we use “Create Advanced Stub (for current Testobject)”.

Tessy has correctly identified that get_job() returns a value to the test object. This is indicated by the passing direction IN for get_job().

**Excursus:**
If get_job() would take parameters, these parameters would also be listed in the interface and would have the passing direction OUT (at least for scalar data types).

**End of excursus**

Please note:
The icon for get_job() has changed. The current icon indicates that Tessy will create an advanced stub for get_job().
2.2 Specify Sequence of Return Values

You can now specify sequences for return values in the TDE perspective.

Because get_job() returns values back to the test object dispatch1(), the return values of get_job() are listed in the Inputs section of the test data.

By using the curly braces '{' and '}', you can specify a sequence of values to be returned by get_job() to the test object dispatch1().

Test case #1 is finalized by the value 0 in the sequence.

Test case #2 is (prematurely) finalized by the (unexpected) value 5 in the sequence. Because we have specified the expected value to be 5, the test case #2 passes, in spite of the premature end.

However, the following message is displayed in the Console view of the TDE perspective.

If a stub function is called more often than values are specified inside the curly braces, the last value inside the curly braces is used repeatedly for the additional calls. (It is not possible to demonstrate this with the example at hand.)

Excursus:
If get_job() would take parameters, these parameters would be listed in the Outputs section and it would be possible to specify a sequence of expected values by using curly braces (analogous to the input values). Tessy would check actual values vs. expected results and if there would be a difference, the test would fail.

End of excursus
3  Call Trace

Tessy is able to check both how often stub functions are called and the order of the calls. The expected result is specified in the Call Trace view of the TDE perspective.

If a call trace is not as expected, the respective test case will fail.

The call trace result is included in the test report.

---

**Fig. 13** Initially the call trace is ignored, i.e. it will not affect the pass/fail result of a test case

You can assemble an expected call trace by dragging&dropping “Available Functions” (right hand part of the Call Trace view) to the “Expected Calls” (left hand part of the Call Trace view).

**Fig. 14** The expected call trace for the first test case, created by drag&drop
After re-execution of the two test cases, the first test case still passes, because the call trace occurred as specified. However, the second test case fails, because the call trace did not occur as specified.
Fig. 18 Call Trace view: The call trace for the second test case has not occurred as specified; it ended prematurely
4 The Test Report

Below are the relevant excerpts from the test report.

Fig. 19 Excerpt from the test report: The test result of the first test case

Fig. 20 Excerpt from the test report: The test result of the second test case
5 Specifying Test Cases

In the example at hand, the test cases were specified with the intention to demonstrate the interaction between the function under test (i.e. dispatch1()) and the stub function for the called subroutine (i.e. get_job()).

However, a more systematically test case specification according to the Classification Tree Method is recommended. The resulting test cases should test the behavior of dispatch1() in the first place, and not its interaction with get_job().

![Classification Tree Editor - (dispatch.cte)](image)

Fig. 21 Test case specification according to the Classification Tree Method

This could be a systematic specification of the test cases according to the Classification Tree Method.

6 Versions

The curly braces as mean to specify vectors of test data for advanced stub functions was introduced in Tessy V3.0.14 and was improved in Tessy V3.0.20.

This document was written using TESSY V3.1.1.

All examples were executed using the Gnu C compiler delivered with Tessy.
7 The Author

Frank Büchner studied Computer Science at the Technical University of Karlsruhe (TH). Since graduating, he has spent more than twentyfive years working in different positions in the field of embedded systems. Over the years, he specialized in the testing of embedded software and passes on his expertise regularly on congresses and seminars.

Frank Büchner is currently employed at Hitex Development Tools GmbH, Karlsruhe.

Frank Büchner
Dipl.-Inform.
Hitex Development Tools GmbH
Greschbachstr. 12
76229 Karlsruhe
+(49) 721 / 9628-125
frank.buechner@hitex.de

Any comments on this paper are very much appreciated.