

Using the Test Driver Communication Tests

Abstract

This document describes the usage of the built-in tests for the target communication of the TESSY test driver. These tests are meant for testing new adaptations of compiler/target environments. They check the transfer of test data to the target as well as the transfer of data back from the target. The tests may be run after adaptation or modification of a specific compiler/target environment.

Please note: When using TESSY within a safety related environment, it is strongly recommended to run the driver communication tests in order to verify the correct behaviour of the communication layer implementation for the specific target that will be used. The tests should be run after initial project setup or after any changes of the environment settings.

It is **not** recommended to have the tests activated all the time, because this consumes much more execution time and much more memory when running tests.

Table of contents

1	Introduction.....	2
1.1	Test driver communication layers.....	2
1.2	Adaptation to the target environment	2
2	Activating the test driver tests.....	3
3	Troubleshooting.....	5
3.1	Selecting individual tests	5
3.2	Reducing the transfer buffer size.....	5
4	Advanced features used primarily by the TESSY support.....	7
4.1	Check correct error handling	7
4.2	Output the actual call trace of major communication functions	7
4.3	Output actual data type sizes	8

1 Introduction

The test driver communication tests may be enabled using module attributes as described below. They check the correct transfer of test data and test results to and from the target device.

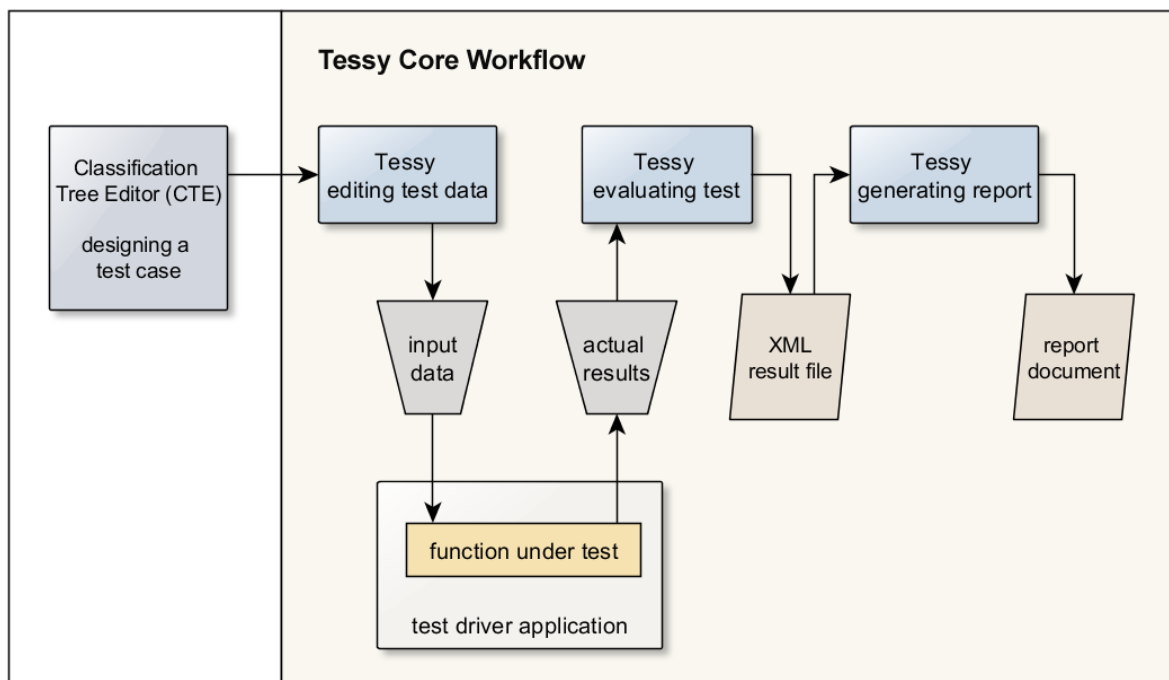
1.1 Test driver communication layers

TESSY uses a dedicated communication channel that is implemented with multiple layers for transfer of test data to and test results back from the target platform. The lowest level layer implements the basic connection to the target debugger which may be based on file I/O or a streaming protocol depending on the communication capabilities of the target debugger.

The next higher level provides transfer of data with specific sizes (e.g. 8 bit, 16 bit, 32 bit, signed and unsigned) and the top level provides the conversion of values as stored within the test database into the raw value that will be transferred to and from the target.

1.2 Adaptation to the target environment

When using testing tools in a safety related environment, it is necessary to qualify these tools according to the respective development standard. The core workflow of TESSY as shown in the picture below has been qualified according to ISO 26262 and IEC 61508 using the GNU/GCC compiler and the respective communication layer implementation.



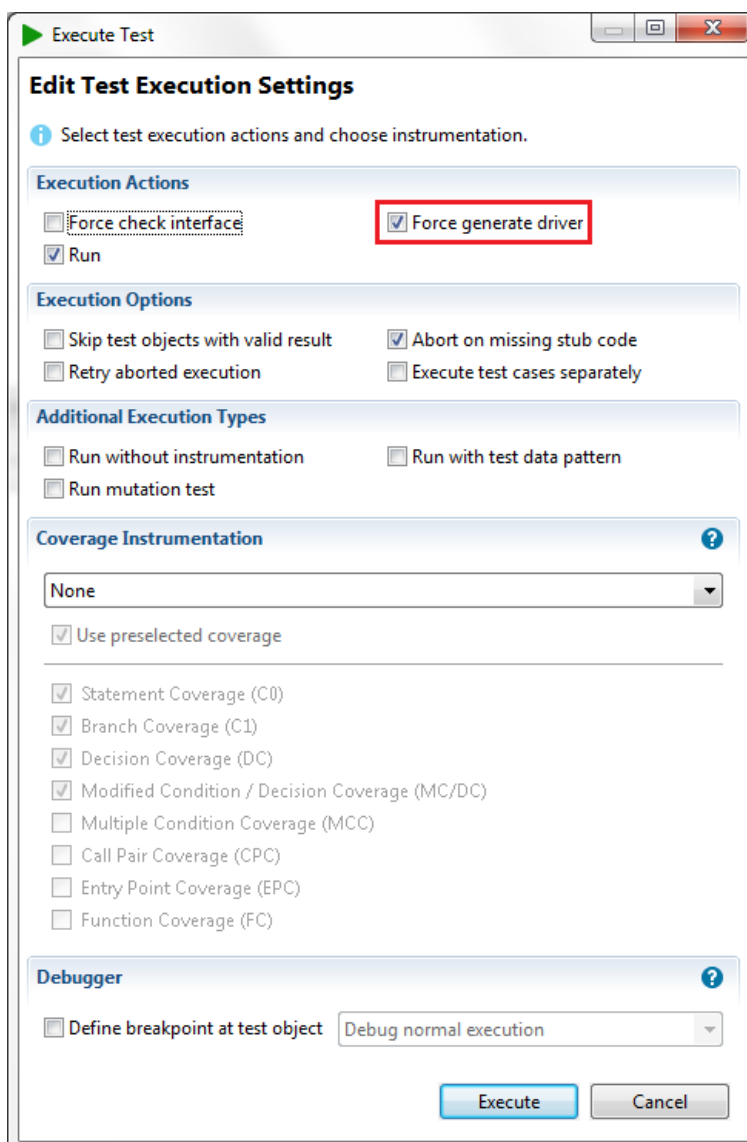
When running tests on a specific target platform, adaptations of compiler options and target debugger settings may be needed within the respective target environment. The verification of the TESSY core workflow covers tests conducted on a Windows host system using the GNU GCC compiler. In order to verify the transmission of test

data and expected results to and from the target device, it is recommended to run the test driver communication tests using the adapted target environment that shall be used for all testing activities (i.e. your specific target environment).

2 Activating the test driver tests

In order to activate all tests concerning sending and receiving of data, set the module attribute **Check Driver Layers** to the value of `0x37`. Please refer to chapter 3.1 for further details.

You need to select at least one test case and rebuild the test driver (i.e. select **Force Generate Driver** within the **Execute Test** dialog) in order to execute the tests:



The result of the test driver communication tests will be printed to the messages window of TESSY. Please check that all tests yield a passed result as show in the messages below (this is a sample test run with UDE debugger and one test case selected):

```

+-----+
| Execute test |
+-----+

Starting UDE...

[checkLayerOne] Send data to the target ...
[checkLayerOne] tslow_put_buffer(uint8[], 2) succeeded.
[checkLayerOne] tslow_put_buffer(uint8[], 4) succeeded.
[checkLayerOne] tslow_put_buffer(uint8[], 8) succeeded.
[checkLayerOne] tslow_put_buffer(uint8[], filled_up_connection_buffer) succeeded.
[checkLayerOne] tslow_put_buffer(uint8[], TS_BUFFER_SIZE - CHECK_SIZE_OF_ONE_ITEM) succeeded.
[checkLayerOne] tslow_put_buffer(uint8[], TS_BUFFER_SIZE + CHECK_SIZE_OF_ONE_ITEM) succeeded.

[checkLayerOne] Receive data from the target ...
[checkLayerOne] tslow_get_buffer(uint8[], 2) succeeded.
[checkGetBuffer] checkuint8Buffer(uint8[], 2) succeeded.
[checkLayerOne] tslow_get_buffer(uint8[], 4) succeeded.
[checkGetBuffer] checkuint8Buffer(uint8[], 4) succeeded.
[checkLayerOne] tslow_get_buffer(uint8[], 8) succeeded.
[checkGetBuffer] checkuint8Buffer(uint8[], 8) succeeded.
[checkLayerOne] tslow_get_buffer(uint8[], filled_up_connection_buffer) succeeded.
[checkGetBuffer] checkuint8Buffer(uint8[], filled_up_connection_buffer) succeeded.
[checkLayerOne] tslow_get_buffer(uint8[], TS_BUFFER_SIZE - CHECK_SIZE_OF_ONE_ITEM) succeeded.
[checkGetBuffer] checkuint8Buffer(uint8[], TS_BUFFER_SIZE - CHECK_SIZE_OF_ONE_ITEM) succeeded.
[checkLayerOne] tslow_get_buffer(uint8[], TS_BUFFER_SIZE + CHECK_SIZE_OF_ONE_ITEM) succeeded.
[checkGetBuffer] checkuint8Buffer(uint8[], TS_BUFFER_SIZE + CHECK_SIZE_OF_ONE_ITEM) succeeded.

[checkLayerTwo] Send data to the target ...
[checkLayerTwo] bufferedWrite(uint8[], 2).
[checkLayerTwo] bufferedWrite(uint8[], 4).
[checkLayerTwo] bufferedWrite(uint8[], 8).
[checkLayerTwo] bufferedWrite(uint8[], filled_up_connection_buffer).
[checkLayerTwo] bufferedWrite(uint8[], TS_BUFFER_SIZE - CHECK_SIZE_OF_ONE_ITEM).
[checkLayerTwo] bufferedWrite(uint8[], TS_BUFFER_SIZE + CHECK_SIZE_OF_ONE_ITEM).

[checkLayerTwo] Receive data from the target ...
[checkLayerTwo] readFromBuffer(uint8[], 2).
[checkGetBuffer] checkuint8Buffer(uint8[], 2) succeeded.
[checkLayerTwo] readFromBuffer(uint8[], 4).
[checkGetBuffer] checkuint8Buffer(uint8[], 4) succeeded.
[checkLayerTwo] readFromBuffer(uint8[], 8).
[checkGetBuffer] checkuint8Buffer(uint8[], 8) succeeded.
[checkLayerTwo] readFromBuffer(uint8[], filled_up_connection_buffer).
[checkGetBuffer] checkuint8Buffer(uint8[], filled_up_connection_buffer) succeeded.
[checkLayerTwo] readFromBuffer(uint8[], TS_BUFFER_SIZE - CHECK_SIZE_OF_ONE_ITEM).
[checkGetBuffer] checkuint8Buffer(uint8[], TS_BUFFER_SIZE - CHECK_SIZE_OF_ONE_ITEM) succeeded.
[checkLayerTwo] readFromBuffer(uint8[], TS_BUFFER_SIZE + CHECK_SIZE_OF_ONE_ITEM).
[checkGetBuffer] checkuint8Buffer(uint8[], TS_BUFFER_SIZE + CHECK_SIZE_OF_ONE_ITEM) succeeded.

[checkLayerThree] Send numbers to the target ...
[checkLayerThree] fill up connection buffer ..... done.
[checkLayerThree] tstcomm_put_char(int8 ch).
[checkLayerThree] tstcomm_put_uchar(uint8 ch).
[checkLayerThree] tstcomm_put_short(int16 number).
[checkLayerThree] tstcomm_put_ushort(uint16 number).
[checkLayerThree] tstcomm_put_long(int32 number).
[checkLayerThree] tstcomm_put_ulong(uint32 number).

[checkLayerThree] Receive numbers from the target ...
[checkLayerThree] receiving filled up connection buffer ..... done.
[checkLayerThree] tstcomm_get_char() succeeded.
[checkLayerThree] tstcomm_get_uchar() succeeded.
[checkLayerThree] tstcomm_get_short() succeeded.
[checkLayerThree] tstcomm_get_ushort() succeeded.
[checkLayerThree] tstcomm_get_long() succeeded.
[checkLayerThree] tstcomm_get_ulong() succeeded.
Test needed 17269 milliseconds

Creating XML result file
Test Execution needed 19656 milliseconds

```

3 Troubleshooting

In case of any problems when running the tests, you may switch on or off individual tests. It may also be necessary to reduce the buffer size that TESSY uses to transfer data to and from the target (i.e. the attribute **Buffer Size**).

If you still fail to execute the test, please contact Razorcat technical support for assistance.

3.1 Selecting individual tests

The value of the **Check Driver Layers** attribute is bit-masked with the following contents:

- 0x10 Send data tests
- 0x20 Receive data tests
- 0x01 Layer one tests
- 0x02 Layer two tests
- 0x04 Layer three tests

You should run all receive data tests first (which is the value **0x27**) and if these tests are successful, you can run all tests (which is the value **0x37**). If you set all bits (i.e. the value **0xff**), this has the same effect as running with **0x37**.

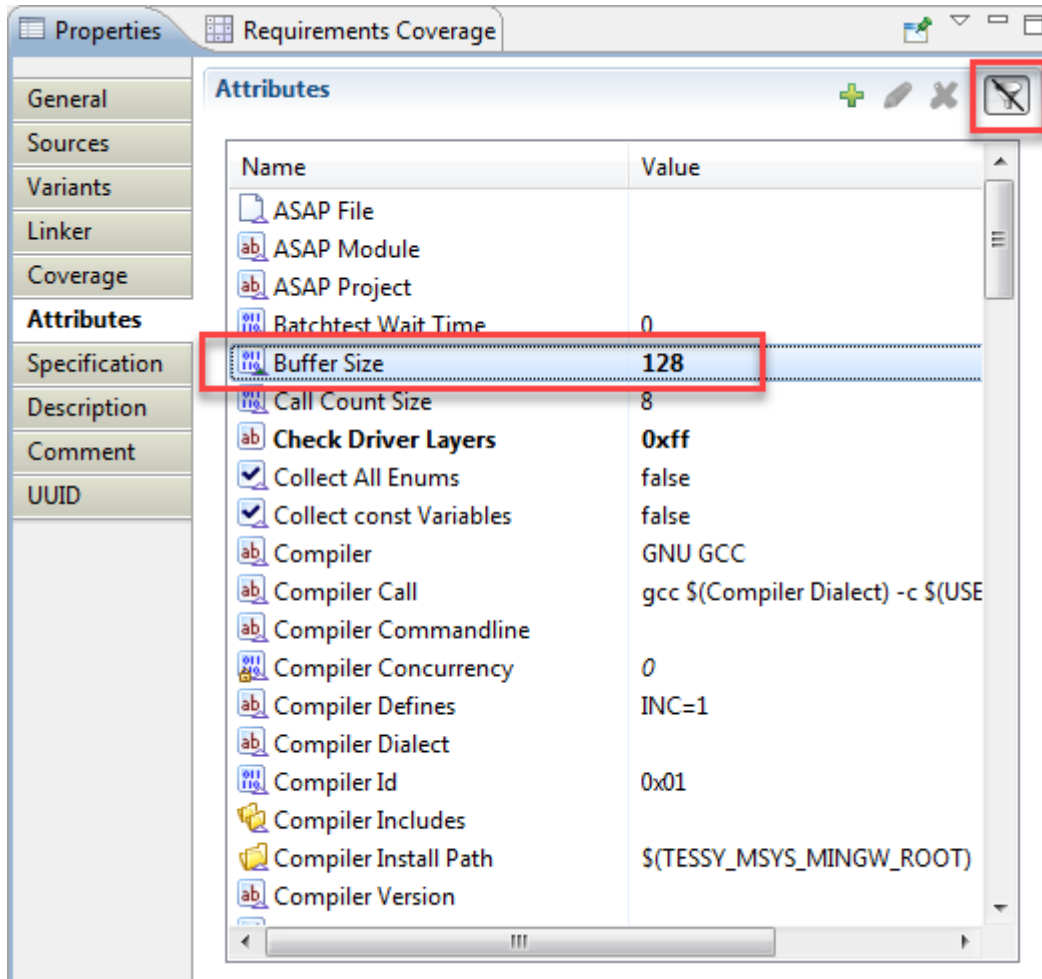
3.2 Reducing the transfer buffer size

The test driver communication tests are using their own local variables for storing test data transferred to the target. Some of these local variables are stored on the stack (i.e. they are defined in the local scope of a function) and their size depends on the **Buffer Size** attribute used for the communication. Since the buffer size is usually as big as possible for a given target, you might need to decrease the buffer size in order to run the test driver communication tests successfully.

Please note: If the tests fail for any reason (e.g. an error message within TESSY or just any exception within the target debugger), you should reduce the **Buffer Size** attribute and execute the tests again.

TESSY Application Notes

As a rule of thumb, you should have the **Buffer Size** not greater than **128** for running the test driver communication tests. Within the properties view of the module, toggle the filter button to have all available attributes being shown:



This will show all attributes for the current environment as listed within TEE. Here you may adjust the **Buffer Size** attribute for the current module only.


```
<- tussy_execute_task()
-> tussy_end_task()
<- tussy_end_task()
-> tussy_start_task()
  -> readFromBuffer()
  <- readFromBuffer()
<- tussy_start_task()
-> tussy_execute_task()
<- tussy_execute_task()
-> tussy_end_task()
```

The example above was invoked by setting the mask to 0x211. Normally it is better to use the mask **0x221** because the call trace is sent in two parts. The first part is sent right after the communication to the target is established while the second part is sent right before the communication is closed.

-> means that the process counter is at the beginning of the function,

<- means that the process counter is right at the end of the function.

The default call trace recording size is 64. You may change this value by TEE expert mode attribute **Check Call Trace Size**. *The value must not be larger than 254!*

4.3 Output actual data type sizes

The mask **0x421** outputs the actual sizes (determined by “sizeof()”) of void *, function pointer, char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, long long, unsigned long long, float, double, and long double. This feature might be helpful to compare the resulting values with the type table. An example is shown below.

[checkLayerOne] Receiving basic data type sizes in bytes succeeded.

```
void pointer      : 4
function pointer  : 4
char              : 1
unsigned char     : 1
short            : 2
unsigned short    : 2
int              : 4
unsigned int      : 4
long             : 4
unsigned long     : 4
long long        : 8
unsigned long long : 8
float            : 4
double           : 8
long double      : 8
```