Quick Start: Using Testkit-lite in Testing

Contents:

1. Introduction

Testkit-Lite works a test runner to automatically run testing. With this test runner, you can run test cases for either middleware components or WebAPIs. This quick-start guide will explain you how to write and structure a test case to be run by Testkit-Lite.
For those who rather have examples than explanations, you can directly check some test cases examples.

2. Testkit-Lite Test Descriptor

For Testkit-Lite to know what test cases are, you need to write a test descriptor file. It's a simple XML file telling how to run test cases. Here is a test descriptor example containing a test case named "HelloWorldTest".

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name=" HelloWorldSuite">
  <set name="HelloWorldSet">
   <testcase component=”HelloWorldComponent" execution_type="auto"
purpose=”Test HelloWorld” id="HelloWorldTest">
     <description>
       <test_script_entry test_script_expected_result="0"
timeout="90">/PATH/TO/HelloWorld
         </test_script_entry>
     </description>
  </testcase>
 </set>
</suite>
```

- **<suite>** and **<set>**

Both of **<suite>** and **<set>** functions as container element in test descriptor. **<suite>** is the Root element of a test descriptor file, which is parent of one or more **<set>** element, which in turn can contain one or more **<testcase>** element as children.

- **<testcase>**

  A test case is defined in **<testcase>** element, which describes its key information, e.g. ID, targeting component, test purpose, pre-conditions, and the most important part, test steps and expected results. For an auto test case, you need to specify "**auto**" as the value of **execution_type**, and give at least one <**test_script_entry>** element to tell which commands to run and with what arguments. Be aware that you need give the absolute path of test scripts/programs/HTML web test page, if it is not accessible with system default **PATH** environment variable. Testkit-Lite verdicts test result to PASS or FAILURE by comparing the real result with value of attribute **test_script_expected_result.**

- <**test_script_entry>**

  <**test_script_entry>** supports below two types of commands:

  - Executable programs or scripts developed in your favor programming/scripting languages. E.g. Python, C/C++, JAVA.
  - HTML web test page. It can either directly embed JS test code in page content, or just has links to separated files containing your JS test code.

  A test descriptor can contain as many test cases as you want, and organized in related test sets, allowing batch execution of all test cases in one test cycle, and testing different test sets. You just need to update the test descriptor when more new test cases are ready.

3. Testkit-Lite Constraints

Testkit-Lite test runner has to make some assumptions about your test cases to reduce the complexity of running them. There are a couple of limitations to be aware of:

a. By default, Testkit-Lite uses system built-in normal account to run your test executables. It is a non-root account.

b. The test script, grogram or HTML web test page given in <**test_script_entry>** should be accessible with system default PATH environment variable. If not, please give full path there to make it accessible by TestKit-Lite.

c. For running **WebAPI** test cases, Testkit-Lite requires a **Web-Runtime Environment** to load your web test pages. A utility tool, **WRTLauncher,** is available for you to easily launch and run WebAPI test cases.

   WRTLauncher takes the name of your widget achieve as input. To launch WebAPI testing:

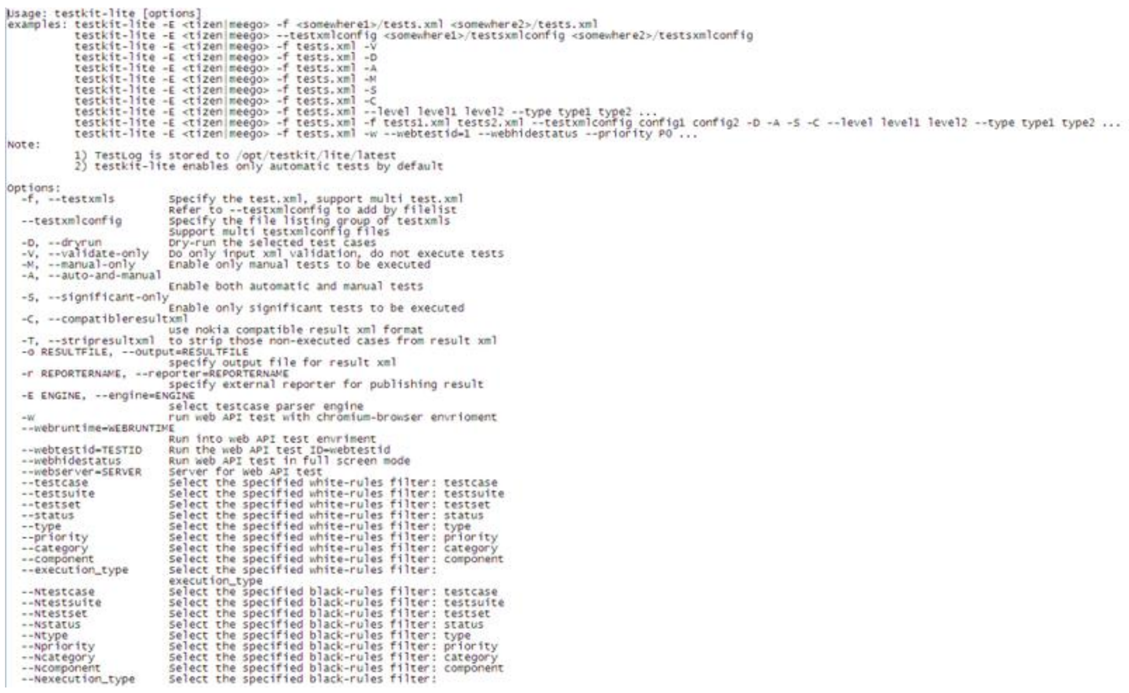
```
$ WRTLauncher  <widget_name>
```

4. Running the Tests

a. Check Testkit-Lite has already been installed in target test device and functions. In terminal, run

```
$ testkit-lite   --help
```

Testkit-lite help information should be printed out on screen



```
Usage: testkit-lite [options]
examples: testkit-lite -E <tizen|meego> -f <somewhere1>/tests.xml <somewhere2>/tests.xml
          testkit-lite -E <tizen|meego> --testxmlconfig <somewhere1>/testsxmlconfig <somewhere2>/testsxmlconfig
          testkit-lite -E <tizen|meego> -f tests.xml -V
          testkit-lite -E <tizen|meego> -f tests.xml -D
          testkit-lite -E <tizen|meego> -f tests.xml -A
          testkit-lite -E <tizen|meego> -f tests.xml -M
          testkit-lite -E <tizen|meego> -f tests.xml -S
          testkit-lite -E <tizen|meego> -f tests.xml -C
          testkit-lite -E <tizen|meego> -f tests.xml --level level1 level2 --type type1 type2 ...
          testkit-lite -E <tizen|meego> -f tests.xml -f tests1.xml tests2.xml --testxmlconfig config1 config2 -D -A -S -C --level level1 level2 --type type1 type2 ...
          testkit-lite -E <tizen|meego> -f tests.xml -w --webtestid=1 --webhidestatus --priority P0 ...
Note:
          1) TestLog is stored to /opt/testkit/lite/latest
          2) testkit-lite enables only automatic tests by default

Options:
  -f, --testxmls        Specify the test.xml, support multi test.xml
                        Refer to --testxmlconfig to add by filelist
  --testxmlconfig       Specify the file listing group of testxmls
                        Support multi testxmlconfig files
  -D, --dryrun          Dry-run the selected test cases
  -V, --validate-only   Do only input xml validation, do not execute tests
  -M, --manual-only     Enable only manual tests to be executed
  -A, --auto-and-manual
                        Enable both automatic and manual tests
  -S, --significant-only
                        Enable only significant tests to be executed
  -C, --compatibleresultxml
                        use nokia compatible result xml format
  -T, --stripresultxml  to strip those non-executed cases from result xml
  -o RESULTFILE, --output=RESULTFILE
                        specify output file for result xml
  -r REPORTERNAME, --reporter=REPORTERNAME
                        specify external reporter for publishing result
  -E ENGINE, --engine=ENGINE
                        select testcase parser engine
  -w                    run web API test with chromium-browser envrioment
  --webruntime=WEBRUNTIME
                        Run into web API test envriment
  --webtestid=TESTID    Run the web API test ID=webtestid
  --webhidestatus       Run web API test in full screen mode
  --webserver=SERVER    Server for web API test
  --testcase            Select the specified white-rules filter: testcase
  --testsuite           Select the specified white-rules filter: testsuite
  --testset             Select the specified white-rules filter: testset
  --status              Select the specified white-rules filter: status
  --type                Select the specified white-rules filter: type
  --priority            Select the specified white-rules filter: priority
  --category            Select the specified white-rules filter: category
  --component           Select the specified white-rules filter: component
  --execution_type      Select the specified white-rules filter:
                        execution_type
  --Ntestcase           Select the specified black-rules filter: testcase
  --Ntestsuite          Select the specified black-rules filter: testsuite
  --Ntestset            Select the specified black-rules filter: testset
  --Nstatus             Select the specified black-rules filter: status
  --Ntype               Select the specified black-rules filter: type
  --Npriority           Select the specified black-rules filter: priority
  --Ncategory           Select the specified black-rules filter: category
  --Ncomponent          Select the specified black-rules filter: component
  --Nexecution_type     Select the specified black-rules filter:
```

b. Deploy to target test device your test descriptor file, related test scripts/programs/HTML web pages, and dependency files or test data

c. Run test cases. Examples here just show introductory usage of Testkit-Lite. Please navigate Testkit-Lite User Guide to know more options and usages.

- To run **Non-WebAPI** test cases:

```
$ testkit-lite   -f /PATH/TO/<test_descriptor_file>.xml
```

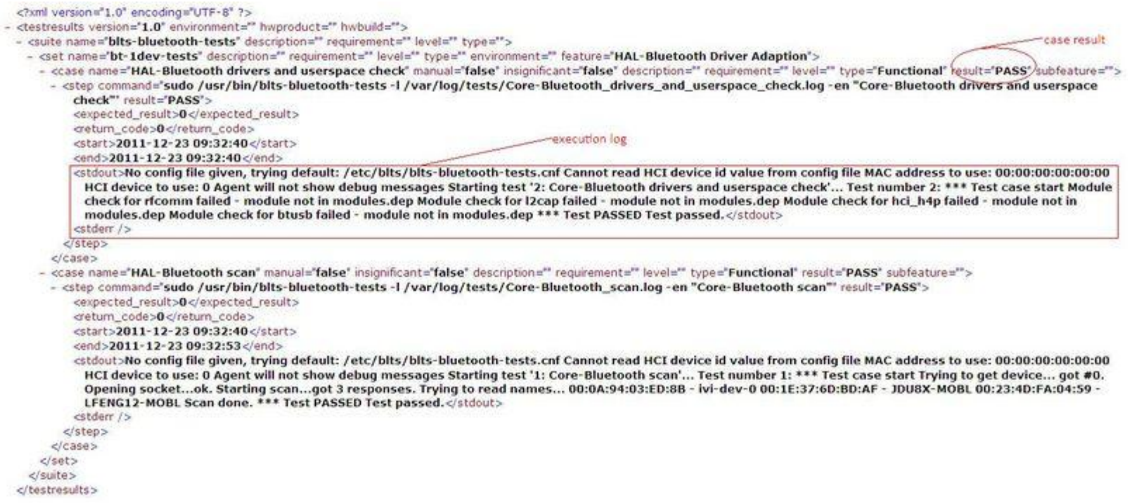
- To run **WebAPI** test cases:

```
$ testkit-lite  -e  "WRTLauncher  <widget _name>" -f
/PATH/TO/<test_descriptor_file>.xml
```

5. Check Test Reports

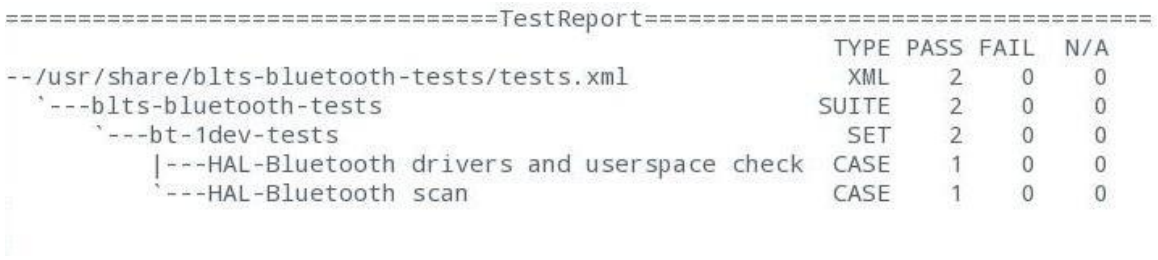
Test reports in both of **XML**, **Text** format are created by Testkit-Lite and located under /opt/testkit/lite/latest after finishing test execution without exceptions. Here are examples of each type of test report.

- **XML Test Report**



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <testresults version="1.0" environment="" hwproduct="" hwbuild="">
  - <suite name="blts-bluetooth-tests" description="" requirement="" level="" type="">
    - <set name="bt-1dev-tests" description="" requirement="" level="" type="" environment="" feature="HAL-Bluetooth Driver Adaption">
      - <case name="HAL-Bluetooth drivers and userspace check" manual="false" insignificant="false" description="" requirement="" level="" type="Functional" result="PASS" subfeature="">
        - <step command="sudo /usr/bin/blts-bluetooth-tests -l /var/log/tests/Core-Bluetooth_drivers_and_userspace_check.log -en "Core-Bluetooth drivers and userspace
            check"" result="PASS">
            <expected_result>0</expected_result>
            <return_code>0</return_code>
            <start>2011-12-23 09:32:40</start>
            <end>2011-12-23 09:32:40</end>
            <stdout>No config file given, trying default: /etc/blts/blts-bluetooth-tests.cnf Cannot read HCI device id value from config file MAC address to use: 00:00:00:00:00:00
              HCI device to use: 0 Agent will not show debug messages Starting test '2: Core-Bluetooth drivers and userspace check'... Test number 2: *** Test case start Module
              check for rfcomm failed - module not in modules.dep Module check for l2cap failed - module not in modules.dep Module check for hci_h4p failed - module not in
              modules.dep Module check for btusb failed - module not in modules.dep *** Test PASSED Test passed.</stdout>
            <stderr />
          </step>
        </case>
      - <case name="HAL-Bluetooth scan" manual="false" insignificant="false" description="" requirement="" level="" type="Functional" result="PASS" subfeature="">
        - <step command="sudo /usr/bin/blts-bluetooth-tests -l /var/log/tests/Core-Bluetooth_scan.log -en "Core-Bluetooth scan"" result="PASS">
            <expected_result>0</expected_result>
            <return_code>0</return_code>
            <start>2011-12-23 09:32:40</start>
            <end>2011-12-23 09:32:53</end>
            <stdout>No config file given, trying default: /etc/blts/blts-bluetooth-tests.cnf Cannot read HCI device id value from config file MAC address to use: 00:00:00:00:00:00
              HCI device to use: 0 Agent will not show debug messages Starting test '1: Core-Bluetooth scan'... Test number 1: *** Test case start Trying to get device... got #0.
              Opening socket...ok. Starting scan...got 3 responses. Trying to read names... 00:0A:94:03:ED:8B - ivi-dev-0 00:1E:37:6D:BD:AF - JDU8X-MOBL 00:23:4D:FA:04:59 -
              LFENG12-MOBL Scan done. *** Test PASSED Test passed.</stdout>
            <stderr />
          </step>
        </case>
      </set>
    </suite>
  </testresults>
```

- **Text Test Report**



```
==================================TestReport===================================
                                                           TYPE PASS FAIL  N/A
--/usr/share/blts-bluetooth-tests/tests.xml                 XML    2    0    0
  `---blts-bluetooth-tests                                SUITE    2    0    0
      `---bt-1dev-tests                                     SET    2    0    0
          |---HAL-Bluetooth drivers and userspace check    CASE    1    0    0
          `---HAL-Bluetooth scan                           CASE    1    0    0
```