MkECLI Tester User Guide

MKE API C++ Client Tester v1.0

Magik Eye Inc.

COLLABORATORS			
	TITLE : MkECLI Tester User Guide		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Magik Eye Inc.	Mar 2021	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
2103-01-EN-00	Mar 2021		MEI

Contents

1 Introduction

This document describes MkECLI Tester, a tool for testing `libmkeclient` [?].

2 Overview of the Used Terms

- **Scenario**: a single testing procedure that can contain one or more tests but should be constrained to test a specific functionality. Each scenario is stored in its own `test_*.cpp` file.
- **Assertion**: a single evaluation check, e.g.: `CHECK(a==b)`
- **Test_case**: same as scenario
- **Section**: one/any of GIVEN/WHEN/THEN blocks.
- **BDD** (*Behaviour Driven Development*) - the testing syntax used in Tester (more info). The syntax looks as follows:

```
SCENARIO( "scenario_name" ) {
  GIVEN( "something" ) {
    WHEN( "something" ) {
      THEN( "something" ) {
        CHECK( testing_condition )
      }
    }
  }
}
```

3 Dependencies

- Catch2 library: already included in the code base as a single header file (`catch.hpp`).

4 Compilation

In order to compile the tester tool, provide the `-DMKECLI_TESTER=ON` CMake option during the `libmkeclient` compilation, *e.g.*:

```
mkdir build && cd build
cmake -DMKECLI_TESTER=ON ..
make
```

5 How to Run

All tests can be run by simply executing the `mkecli_tester` binary:

```
./mkecli_tester
```

The above command will start testing all the available scenarios and it will output errors only. It can take up to a few minutes to finish and it will output summary statistics upon completion. An illustrative example of several tests failing:

mkeclient tester output

5.1 Available options

The available options can be explored using the command line parameter `help`:

```
./mkecli_tester --help
```

Most of the available parameters and their handlings are provided by the `Catch2` library. Parameters provided by MkE Client Tester itself are at the end of the list and are denoted by the `MkE:` prefix in the parameter description.

5.2 Setting connection

Currently, the only supported protocol for testing is TCP. The TCP connection can be specified using the following optional parameters:

- `host`: default value is "localhost"
- `port`: default value is "8888"

Examples:

```
./mkecli_tester --host localhost
./mkecli_tester --host 127.0.0.1 --port 8888
```

5.3 Selecting Tests

List of available scenarios can be obtained using the `-l` parameter:

```
./mkecli_tester -l
```

The above command will output a list similar to the following one:

```
All available test cases:
  Scenario: core_async_cascading
  Scenario: core_fast_requests
  Scenario: core_invalid_connection
  Scenario: core_memory_limit
  Scenario: core_multiple_connections
  Scenario: core_offline
```

```
  Scenario: core_stability
  Scenario: core_timeouts
  Scenario: device_info
  Scenario: device_info_async
  Scenario: device_state
  Scenario: device_state_async
  Scenario: firmware_info
  Scenario: firmware_info_async
  Scenario: frame_push_async
  Scenario: frame_push_async_interrupt_by_state_change
  Scenario: frame_push_stop_sync
  Scenario: frame_push_when_already_pushing
  Scenario: get_frame
  Scenario: get_frame_async
  Scenario: get_frame_exceptions
  Scenario: policies_get_async
  Scenario: policies_list_async
  Scenario: policies_set_async
  Scenario: terminate
  Scenario: terminate_async
26 test cases
```

A specific scenario can be executed by providing its name as a parameter:

```
./mkecli_tester "Scenario: device_info"
```

Tests that try to break things (e.g. clog connection with requests) or take more time are disabled by default. These tests can be allowed using the `aggresive` parameter and it is recommended to run them manually one by one so that the other tests are not affected. The list of tests that may get aggresive:

```
./mkecli_tester --aggresive yes "Scenario: core_invalid_connection"
./mkecli_tester --aggresive yes "Scenario: core_memory_limit"
./mkecli_tester --aggresive yes "Scenario: core_multiple_connections"
./mkecli_tester --aggresive yes "Scenario: core_stability"
./mkecli_tester --aggresive yes "Scenario: core_fast_requests"
./mkecli_tester --aggresive yes "Scenario: terminate_by_reboot"
./mkecli_tester --aggresive yes "Scenario:  ↩
   terminate_by_reboot_and_reconnect"
./mkecli_tester --aggresive yes "Scenario: terminate_by_shutdown"
```

5.4 Output Formatting Options

Show all tests (even those successful ones):

```
./mkecli_tester -s
```

Show tests using a compact format, one test per line:

```
./mkecli_tester -r compact
```

Parameters can be combined, *e.g.*:

```
./mkecli_tester --aggresive yes --host localhost --port 8888 -r  ↩
   compact -s "Scenario: device_info"
```

6 Writing Custom Tests

- Copy any existing scenario and save it under new file name, *e.g.*, `test_new_scenario.cpp`
  - To make things organized, do not forget to change the name in the beginning of the file. *e.g.*: SCENARIO("new_scenario"), so that it corresponds to the new filename
- Write tests. Few tips:
  - Get inspiration from other scenarios.
  - Write short and self-descriptive scenarios.
  - Do not over-complicate things so someone else can understand what went wrong with the test as quickly as possible.

7 Bibliography

- [] *MagikEye API v1.0*, 2020, Magik Eye Inc.
- [] *MkECLI User Guide: MKE API C++ Client v1.0*, 2021, Magik Eye Inc.