**Description:**

This Virtual Device aims at simulating a real Device in terms of sending data(in byte[] format) back to Android programmer. Virtual device is using TCP/IP communication protocol instead of BLE in real running environment.

**Virtual Device has following DeviceMode: (ignore this part if you don’t review the code)**

1. Non-testing mode (deviceMode=0): Device data (which can be programmed), such as parameter data, is serialized and deserialized locally through IO. If local data is not available, default values are returned. Non-programmable data will return random values (such as Device Log, Episode, etc.).

2. Espresso testing script mode 1 (deviceMode=1): Fuzzy testing + Semi-random: Espresso retrieves the data and parses it locally, then compares it with the actual UI data. All returned data is randomly generated within a reasonable range.

3. Espresso testing script mode 2 (deviceMode=2): Fuzzy testing + Fully random: Espresso retrieves the data and parses it locally, then compares it with the actual UI data. All returned data is randomly generated without any pattern.

4. Espresso testing script mode 3 (deviceMode=3): Regression testing. Predefined data and expected test results: Espresso does not need to parse the data locally. Each Testcase will have predetermined expected data and results. Espresso should directly compare whether the results are correct.

5. Espresso testing script mode 4 (deviceMode=4): Integration testing. Collaboration of multiple modules, such as popup testing.

6. Espresso testing script mode 5 (deviceMode=5): Testing is aimed at all programmable+retrievable parameters. For example, using Brady, Espresso randomly opens parameter settings, randomly clicks an option. During the program, it records all data on the current UI screen, then retrieves it for comparison. If the data matches, the test passes.

7. Stress/Performance testing (deviceMode=10): Testing the Programmer's performance when faced with a large amount of garbage/illegitimate data.

The DataFlow of Virtual Device:

**TCP Server-Client Interaction and Data Processing Workflow**

* Start
* TCP Server Listens on port 8888
* Client Connects -> Server creates new Thread (Classical BIO Model)
* Thread Creates Controller.ICDDevice instance
* Two types of Incoming data, one is testing environment configuration, another is real data from programmer
* Call DecodingPacket class to decode the packet into: packetHeader(3 bytes) + payload + CRC32
* Packetheader contains: packet length + sequenceNumber + commandId.
* Perform Corresponding I/O operations based on commandId
* Each ICDDevice Process corresponding data
* Server sends encoded packet back to client
* Repeat from Step V. (until client disconnects)
* End