Android Test Tool

Additional instructions

The usage instructions can be found in the README.md file present in the main folder. This file provides additional technical information, for maintenance and continuity purposes.

This tool is provided via an installer. In the installation folder, open setup and follow the steps. The destination folder must be a permission-free folder (e.g. ‘User’ folder, and not ‘Program Files’). Once installed, open the Test Tool application. It uses ADB to communicate with Android Phones. ADB does not come by default during the first installation. The latest version will be downloaded from internet and unzipped, and be available after in the ‘platform-tools’ folder. ADB assigns unique IDs to devices, and those IDs are used when multiple phones are connected with the following syntax:

adb -s <adb\_id> shell “<Enter shell command>”

Python version must be at least **Python** **3.7**, because the dictionaries are insertion-ordered since this version. Without this characteristic, there may theoretically be errors when reading simInfos.csv (not tested), because the order is important for the get\_dictionaries() method of Test\_Tool.py. You also need to install the package ‘mysql.connector’.

The Python file Test\_Tool.py contains the main logic of the program:

* First, it reads the file simInfos.csv, and for each sim will try to get its ADB ID. It will then create 3 dictionaries to establish a correspondance between a phone’s MSISDN, IMSI and ADB ID (those dictionaries are obtained via the function ‘get\_dictionaries()’).
* Then, it will read testsToPerform.csv line by line and execute the function corresponding to the line in the file, with all its parameters.

For moc\_routine() and sms\_routine(), the Phone B field can either be a phone number, or the index of a phone in the list. Considering that phone numbers are at least 3 digits (shortcodes for example), any number that is less than 3 digits will be considered as the index of a phone in simInfos.csv (thus the limitation on 99 entries in simInfos.csv). If this limit needs to be raised to 999 entries, lines 76 and 144 must be changed to:

‘if len(index\_b) > 3:’

in Test\_Tool.py.

For sms\_routine(), the double quote (“) character poses a problem when present in the text to send. For this reason, it is replaced with a single quote (‘) before sending the message (line 143).

For speedtest\_routine(), the test will open the default browser and download a file from [this site](http://ipv4.download.thinkbroadband.com). This site hosts files of a given size (From the website: “These files are made of random data, and although listed as zip files, will appear to be corrupt if you try and open them”). The options available in Test\_Tool.py are 10, 20, 50 and 100MB, and the size is tested at line 199. If you want to add a size, add it here and in the combobox of the WPF interface named “File Size (Speedtest)”. The speed is calculated by writing to a file the time of start and end in the batch script (precision is to the millisecond), and then using Python to convert the timespan into seconds and microseconds using the datetime library. However, Chrome and other browsers cache around 5-10MB of the downloaded files, even when the file is deleted from phone storage. Thus, the speedtest may be imprecise for small files (10 and 20MB).

Every test in the list will output its result to the corresponding log file found in the logs/ folder. It contains information about the parameters of the tests, as well as the date and time of the tests and the results. (Some errors might not be handled properly, need more testing). However, some tests results cannot be retrieved. For example, we cannot know if an SMS has been properly received, or if a web page has been correctly opened in the Data test. These tests require manual checking. The folder also contains a file called ‘SMSdatabaselog.csv’. This file contains information retrieved from the CDR database about the messages sent (Time is UTC time, but time in SMSlog.txt is local time).

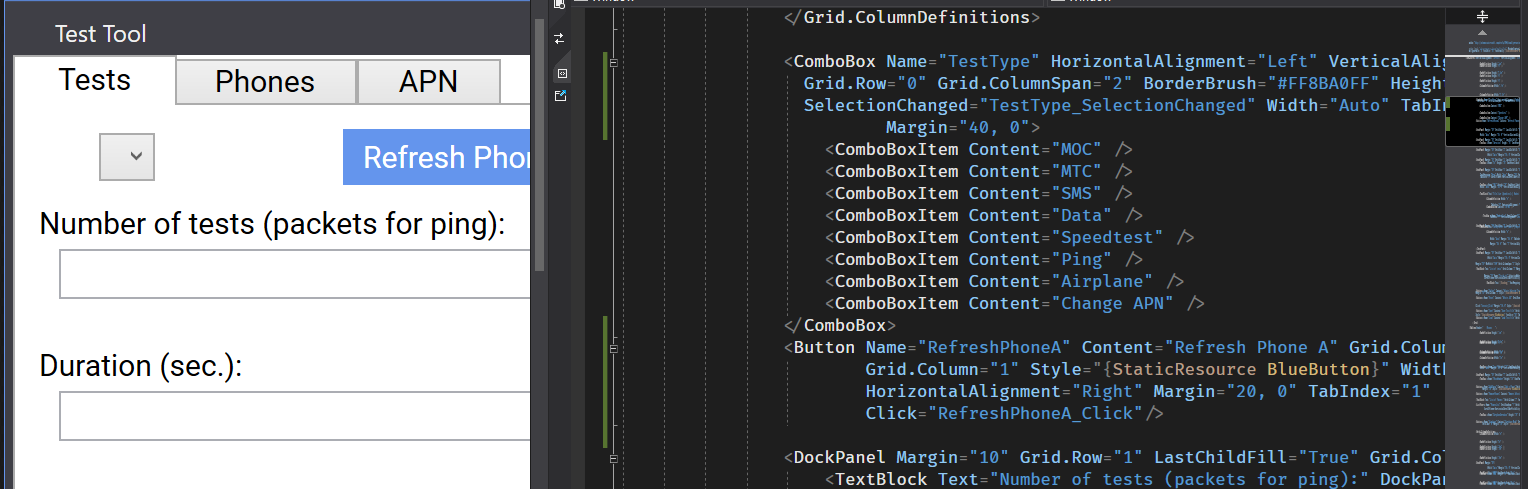
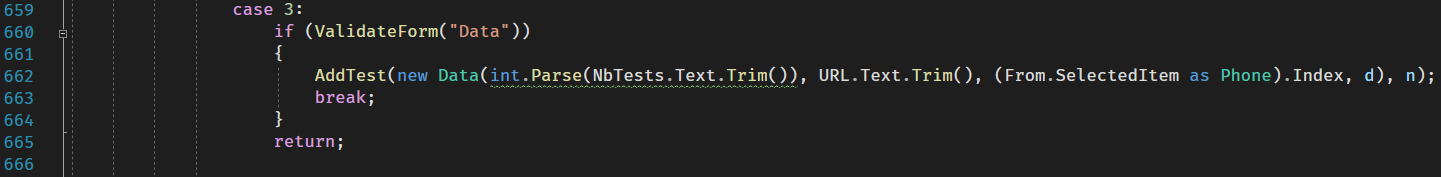
All routines have a timeout defined, the time after which the batch file execution will stop in case a routine is excessively long. This timeout will be written in the logs, and the reasoning behind the timeout duration is explained in each routine of the Test\_Tool.py file. This timeout can obviously be changed, especially for the speedtest routine, where the timeout is set to trigger if the average download speed is inferior to 100kB/s approximately.

If a rooted phone is used, make sure that it has the app [‘SuperSu’](https://supersuroot.org/) installed, and grant SU permission for shell (or go into SuperSu > Settings > Grant access by default). For all phones, do not forget to enable USB debugging in the developer options as explained in README.md.

The WPF code is essentially a practical way to write the tests to testsToPerform.csv, view/add phones to simInfos.csv, and view the available APNs of a particular phone. It provides error checking for all fields, so every test/phone added via the interface should be syntactically correct for Test\_Tool.py to understand.

If a test needs to be added to the list, there are multiple steps:

In WPF/C#:

1. Add it as a ComboBoxItem to the first ComboBox in the Tests tab of MainWindow.xaml.
2. Eventually, add new UI elements if the test requires it.
3. Add a case in the switch in TestType\_SelectionChanged(). This is used to disable UI elements that should not be set for the specific test. The switch operates on SelectedIndex, which is the 0-based index of the test in the list of ComboBoxItems. In this case, set the correct UI elements to be enabled/disabled for the new test. Note that the Delay, Phone A and Repetitions field are always active.
4. Add a class to the solution for the new test. This class should implement the ITest interface (and all of its members). ITest defines the minimal required arguments for a test. Your new test’s class also needs a constructor with all its parameters. For the WriteCsv() method, you should follow the syntax of other tests: a semi-colon-separated string with the name of the test in lower case first.
5. Add a case in the switch in Add\_Click(). This case should have the same structure as the others (see screenshot), and uses the same index as in step 3: It calls the ValidateForm() method, with a string argument specific to the new test (more on this below). If Validate\_Form() returns true, it calls for the AddTest() method, taking as parameter the instantiation of you newly created test object. The parameters are obtained by getting the user input of the corresponding fields. The parameters ‘d’ and ‘n’ are the delay after the test and number of tests to add, respectively.
6. In the ValidateForm() method, add a case using the string you put in argument to ValidateForm() before. In this case, check for user input correctness (see the other tests as examples). If a field is not correct, return false. If all tests pass, break from the switch (it will return true).
7. The test will then be automatically added to the list of tests and written to the tests file. Make sure the WriteCsv() method is correct.

In Python:

1. Open Test\_Tool.py, and a case for your new test in get\_test\_list():

‘elif l[0].lower() == "new\_test\_name":’

where new\_test\_name is the first column in your WriteCsv() method in lowercase.

1. Define a function for your test, taking as parameters ‘index’ (the index of the phone to use), and all the other parameters of your test.
2. Call your method with the corresponding arguments. These arguments are stored in the list ‘l’, in the same order as in your WriteCsv() method. The ‘index’ parameter must be cast to an int in the method call. After your method, call time.sleep(int(l[-1]) if your delay parameter is correctly set to the last element of WriteCsv().
3. Your new function should follow the syntax of the others: define the timeout, try to get the ID of the phone and run the .bat file corresponding to your test.

to = \*\*\*\*\*

num = tuple(number\_to\_imsi.items())[index - 1][0]

try:

id = imsi\_to\_id[number\_to\_imsi[num]]

except:

print(f"Selected phone is not plugged in ({num})", file=sys.stderr)

return

print("\n[{}] Beginning NEW\_TEST routine...\n".format(str(datetime.now().strftime("%H:%M:%S"))))

try:

subprocess.run(["NEW\_TEST.bat", id, param1, param2, ...], timeout=to)

except:

with open("logs\\NEW\_TEST\_LOG.txt", "a") as f:

f.write("[{}] NEW\_TEST unsuccessful (process timed out) (param1: {}, param2: ...)\n\n".format(

str(datetime.now().strftime("%d/%m/%Y-%H:%M:%S")), param1, param2, ...))

In Batch:

1. Create a batch file NEW\_TEST.bat in the same directory as the others.
2. In it, run your adb command and output the result to the log as in the other batch files. The syntax of the adb command should be:

adb -s %1 shell “<Enter your command here>”.

1. Read the other batch file to be coherent with the other tests in your own batch file.