AndroidCAR by Ri.se - user instructions

August 30, 2018

1 Android/CAR by Ri.se

1.1 Overview

Android/CAR by Ri.se is an Android 8.1.1 (API 27) derived Android containing the CAR API. It is intended for use by developers working on host using the Android Studio emulator as target.

1.2 Installing

These instructions will help you install the Android/CAR by Ri.se image to your Android Studio environment. The prefered host operating system is Linux (Ubuntu 16.04 or 14.04) but the *equivalent* set of commands will install to a Windows host. This instnace of Android also use the Java 8 framework version and do not necessarily accept higher versions and definitly *not* lower Java versions.

Note that for the example command line text below the paths to frameworks, tools and workspaces used are those set by the author (who run an Ubuntu 16.04 Linux host) and *will* be different on other's computers. It is also assumed that relevant PATH variables are set appropriatly for access to tools and commands.

1.2.1 Unpack

Make a suitable working directory and unpack the *android-car-<some-date>.zip*.

```
$ pwd
/home/s19694/projects/oil/slask/proj
$ unzip android-car-180829.zip
Archive:  android-car-180829.zip
  inflating: carapi.py
  inflating: examplesim.py
  inflating: propdb.py
  inflating: sgtranslator.py
  inflating: config.ini
  inflating: sgt.ini
  inflating: AndroidCAR by Ri.se - user instructions.pdf
 extracting: rice-android-27.zip
 extracting: MyApplication.zip
```

1.2.2 Install Android image

First, navigate to your Android SDK directory and then to the subordinate *system-images* folder (on Windows it is typically located in the user account AppData\Local folder). Then unzip the *rice-android-27.zip* archive at that location. That would extract an Android image directory and file structure conforming to the Android toolset preferences. It could look something like this:

```
$ cd ~/opt/android/sdk/system-images/
$ pwd
/home/s19694/opt/android/sdk/system-images
$ unzip rice-android-27.zip
Archive:  rice-android-27.zip
   creating: rice-android-27/
   creating: rice-android-27/default/
   creating: rice-android-27/default/x86_64/
  inflating: rice-android-27/default/x86_64/kernel-ranchu
  inflating: rice-android-27/default/x86_64/package.xml
  inflating: rice-android-27/default/x86_64/build.prop
  inflating: rice-android-27/default/x86_64/vendor.img
  inflating: rice-android-27/default/x86_64/userdata.img
  inflating: rice-android-27/default/x86_64/system.img
  inflating: rice-android-27/default/x86_64/encryptionkey.img
  inflating: rice-android-27/default/x86_64/advancedFeatures.ini
  inflating: rice-android-27/default/x86_64/ramdisk.img
$ ls -l
total 963224
drwxrwxr-x  7 s19694 s19694      4096 Aug  8 11:25 ./
drwxrwxr-x 14 s19694 s19694      4096 Jul  6 10:32 ../
drwxrwxr-x  3 s19694 s19694      4096 Aug  7 14:58 rice-android-27/
-rw-rw-r--  1 s19694 s19694 492653609 Aug  7 15:18 rice-android-27.zip
```

Nice.

Now, we need to construct an AVD instance to run in the simulator. The safest way to do that is via the command line, *not* the Android Studio!

```
$ avdmanager create avd -n AndroidCAR -k "system-images;rice-android-27;default;x86_64"
```

You do not wish to "create a custom hardware profile" so just press return if asked. Observe that the "path-like" parameter to the *-k* flag match your path from your *system-images* directory to your Android image. The *AndroidCAR* parameter can be set to whatever you like, it is the (base) name given to the AVD instance and the directory and setup files.

You should now be able to see an AVD instance in Android Studio's AVD manager, but **don't** start it yet! Before starting the emulator we need to provide a proper configuration. When unpacking the base zip-archive there where a *config.ini* file that unfolded at the top. Move that to the AVD directory created by the above command (on Windows it typically reside as a folder .android at the user's account top). Observe that it will overwrite an already present, but incomplete, configuration generated by *avdmanager*.

```
$ pwd
```

```
/home/s19694/projects/oil/slask/proj
$ mv config.ini ~/.android/avd/AndroidCAR.avd/
$ ll ~/.android/avd/AndroidCAR.avd/
total 563768
drwxrwxr-x  2 s19694 s19694      4096 Aug  8 11:28 ./
drwxrwxr-x 10 s19694 s19694      4096 Aug  8 11:27 ../
-rw-rw-r--  1 s19694 s19694       331 Aug  7 15:18 config.ini
-rw-r--r--  1 s19694 s19694 576716800 Aug  8 11:27 userdata.img
```

Now you should be able to start the emulator from the Android Studio interface as you would for any Google provided device.

1.3 Auxillary tools and files

To work with the Android emulator and the CAR API there are a couple of auxillary files and tools:

carapi.py defines a Python 3 interface to the event message mechanism that is used internal to Android to propagate events to the CAR API. Use this interface in a Pyahon based discrete event simulator of your choise (e.g. SimPy) to simulate vehicle mechanics and signals propagating to an app. An example of a simulator, written using SimPy, can be found in file **examplesim.py**.

sgtranslator.py is another Python 3 tool that implement a translator process between Secure-Gateway MQTT messaging of vehicle signals to the Android/CAR emulator. It's perfectly possible to use this tool without the SequreGateway by writing a separate MQTT producer that simulate a signal stream and connect the translator process to it.

Both the above tools use a database defined by **propdb.py** to get details on singal and event data. This is where Android events/properties are declared and defined for use by the carapi and the mappings between events and SequreGateway MQTT topics are set.

sgt.ini is the configuration file for sgtranslator.py.

An example app that works with the examplesim.py simulation is present in **MyApplication.zip**.

These files where unpacked at the initial working directory. Move them to wherever they are needed and enjoy. Further information and documentation on the workings of these tools are available in the respective file.