Data Collection Application Documentation

Overview:

This application collects data from many modalities, these include:

1. Accelerometer (Accelerometer data from the phone, this data excludes gravity. This data changes very rapidly)
2. Air pressure (The air pressure of the environment, some phones have this sensor)
3. Ambient Light (The light from the environment, also collects screen brightness if it is manually set and not automatic)
4. Battery (The battery level of the device)
5. Call state (Whether or not the phone is idle, ringing, or in a phone call)
6. Camera (Front and back images)
7. Connectivity (Wifi, Bluetooth, Cellular)
8. Foreground (List of applications that the user is currently using)
9. Location (GPS)
10. Magnetic data (The strength of magnetic fields of the environment. Used for bearing)
11. Proximity (Whether the phone is near or far from a surface)
12. Rotation (Uses gyroscopic data from the phone to calculate its rotation)
13. Screen state (Whether or not the phone screen is off or on)
14. Temperature (The temperature of the environment, some phones have this sensor)

This application collects this data and uploads the data to an Amazon S3 account for storage. In order to use this feature, make sure that the data collection services are started using the toggle on the top of the activity to switch on all services.

Architecture:

Each modality has a service responsible for collecting its data. There are three ways this data is collected. The first way is to register with the device’s sensor manager to obtain readings from the sensors at a loosely defined sampling rate. The second way is to create timers that poll the phone’s internal state. The third way is to register with third party sources which include the phone’s broadcast intents, and google play services.

The services that start runnables that register with the device’s sensor manager are:

Accelerometer, Air pressure, Ambient Light, Magnetic, Proximity, Rotation, and Temperature.

The services that use timers to poll the phones internal state are:

Battery, Camera, Foreground

The services that register with the phone’s broadcast intents are:

Call state, and Screen state

The service that register with Google Play services is:

Location

Each of these services is responsible for writing the sensor readings into their respective files. Whenever it gets a new reading from the modality, it formats it and writes it into a file. There is only one file for each of the modalities except for the camera. These files are found in /sdcard/Android/data/com.example.hooligan.accelerometerdatadumper/files/<username>\_<timestamp> and can be viewed using the ES File Explorer.

Each modality is associated with a java package which includes the fragment, the fragment interface, the service, and if it registers with the device’s sensor manager then it has a runnable file that registers with the device sensor manager. Otherwise, the functionality for recording samples are found in the service.java file associated with the modality.

The entry point to the application is the SensorDataDumperActivity. This activity is responsible for starting all the services for each modality as well as uploading the directory that contains all the files to the Amazon S3 storage service.

Camera Architecture: Example here: [Example](https://github.com/googlesamples/android-Camera2Basic)

FrontBackCameraService\_2.java is the file that handles the service for capturing front and back images. The timertask is responsible for opening the front camera first and capturing and saving an image and then opening the back camera and capturing and saving the image. The front image and back camera happen sequentially and happen every capturing period. There are several key functions and members that are important to this control flow.

mImageReader is responsible for saving the image once the image has been captured.

mCameraManager is responsible for opening each camera device. Once the opening is successful, it will return to the onOpened() method of mStateCallback. This enables the capturing of the image to be executed and the beginCapture() method is then called.

beginCapture() is responsible for making sure the window size has not been reached and also verifying that the correct filename is associated with the camera device being used (front camera or back camera). It then proceeds to createCaptureRequest().

In createCaptureRequest(), we build the requestBuilder which has options for the capture. The callback is also declared and passed into the method for mCameraCaptureSession.capture(). Once the capture has completed it will return to the callback of captureCallback and the meta data can then be saved about the image.

The images are saved using a class called ImageSaver.java which just writes information into a jpg file.

The meta data is saved using CameraMetaSaver.java and is called when the capture has been completed in the captureCallback. All the meta information is dumped into a file and the type and amount of meta data is dependent upon the device. The nexus 6 includes many fields so that might be a good device to use for obtaining meta information.