IoTNode Users Guide

IoTNode Users Guide
Version 1.0.0, January 2018

http://www.analog.com

Contents

1 Introduction

1.1 Purpose

This document describes the IoTNode Android application for Android Software Development Kit (SDK) versions 19 or greater.

1.2 Scope of this Manual

This document describes how to use the IoTNode Android application. It explains how to get started with the application and its current functionality. The second portion of this will detail how to expand and modify the application to meet custom requirements.

This document is intended for engineers who integrate ADI's device drivers and Bluetooth framework with other software to build a system. This document assumes some familiarity with the Android development environment.

1.3 Acronyms and Terms

ADI	Analog Devices, Inc.
API	Application Programming Interface
BSP	Board Support Pack
SDK	Software Development Kit

1.4 Android Terminology

Activity	A process in the Android app that displays information to the user
Service	A process in the Android app that runs in the background of an Activity
BroadcastReceiver	An Android object setup to handle data sent between Activities and Services
Layout	A visual structure for a user interface, such as the UI for an activity
ListView	A view group that displays a list of scrollable items

2 Product Overview

Analog Devices' (ADI) **IoTNode** app enables a quick and easy way to connect to smart IoT sensor nodes. It allows a user to get localized sensor data from platforms that utilize ADI's advanced sensors and ultra low power technology.

The platforms showcase ADI technologies that sense, measure, interpret, and connect in ways that add integrity and intelligence at the edge, improving the overall reliability, impact, and efficiency of the IoT solution.

Configuration of the platform is performed using this Android application running on an Android phone or tablet using Bluetooth Low Energy (BLE) 5.0 Link between the mobile device and the development platform.

Supported Platform(s):

- EVAL-ADICUP3029
    - The EVAL-ADICUP3029 is an Arduino and PMOD compatible form factor development board that includes Bluetooth and WiFi connectivity. The board uses CrossCore Embedded Studio, an open source Eclipse based Interactive Development Environment (IDE), which can be downloaded free of charge. The platform contains many hardware and software example projects to make it easier for customers to prototype and create connected systems and solutions for Internet of Things (IoT) applications. The EVAL-ADICUP3029 uses the ADuCM3029 ultra low power Arm Cortex-M3 processor as the main device on board. The ADuCM3029 is an integrated mixed-signal microcontroller system for processing, control and connectivity.

3 Installation Components

3.1 Installing IoTNode.apk

This section provides step-by-step instructions on how to install the Android host application onto an Android device. Note: The pictures in this section are from a Samsung Galaxy and may vary slightly from device to device.

3.1.1 Part 1: Copy IoTNode.apk From the Computer to the Android Device.

If you are using a PC:

1. Connect the Android device to a computer using a Micro-USB to USB cable.
2. Open the "Computer" file browser, shown in Figure 1. The device should appear like a drive, for example "Galaxy Tab A" or "SAMSUNG_SM_G930V", under the "Portable Devices" tab.

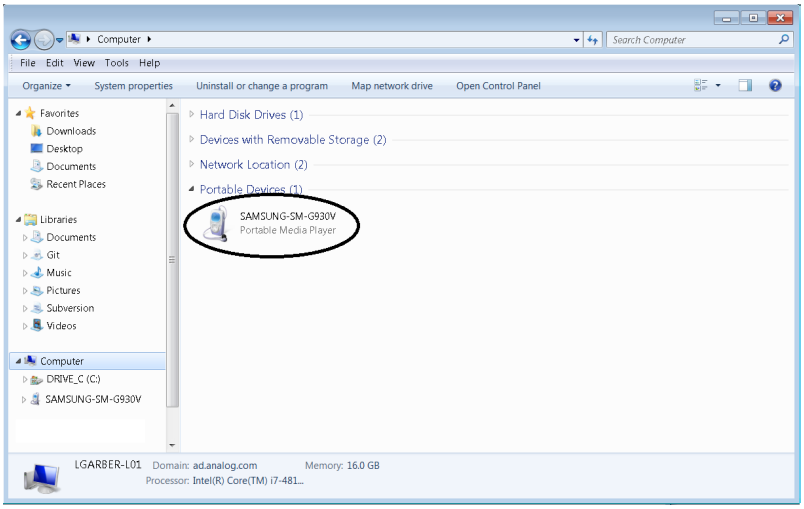


Figure 1. "Computer" File Browser

3. Navigate to the device's "Download" folder, shown in Figure 2.
   Note: If the device appears empty, please go to step a to enable MTP or USB mass storage on the Android device. Otherwise continue on to step 4.

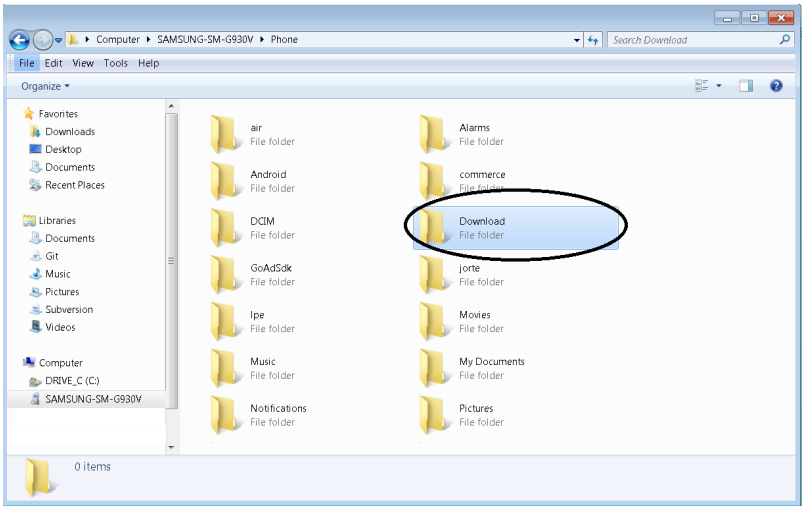


Figure 2. "Download" File Browser

a. On the Android device, pull down the notification menu at the top of the screen. This can be seen in Figure 3. Click on the tab titled "Android System".

Figure 3. Notification Menu to Enable Media Transfer to the Android Device

b. Change the selection to "Transfer files" or "MTP", as shown in Figure 4.

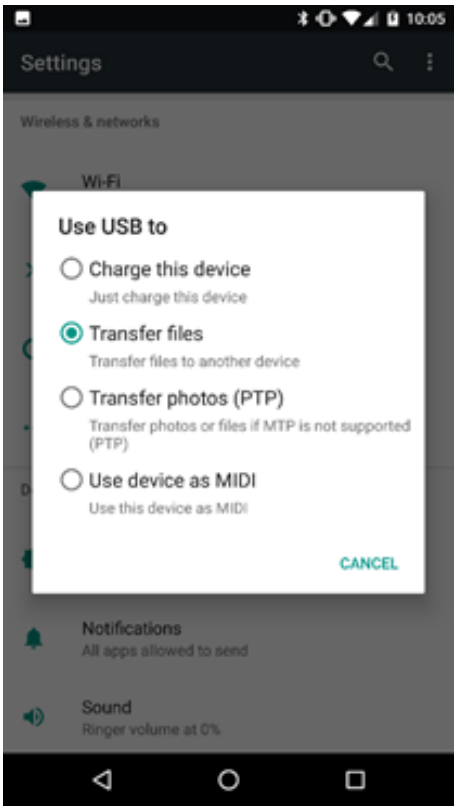


Figure 4. Settings Menu to Enable Media Transfer to the Android Device

c. Now back on the PC, navigate to the "Download" folder of the Android device.

4. Copy IoTNode.apk file to the "Download" folder of the Android device. This can be seen in Figure 5.

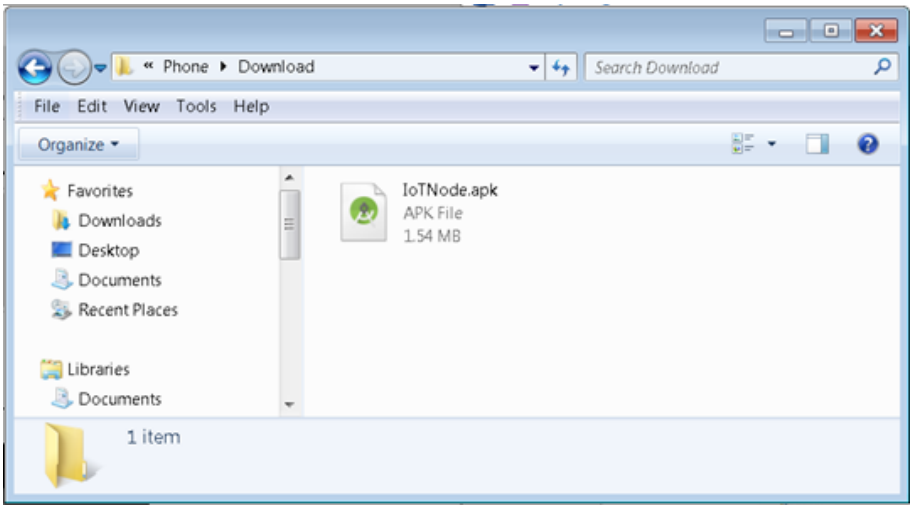


Figure 5. IoTNode.apk Copied into Android Device "Download" Folder

If you are using a Mac:

1. With the Android device **not** plugged in to the Mac, download Android file transfer from https://www.android.com/filetransfer/.
2. Open androidfiletransfer.dmg.
3. You should see a pop-up that looks like the Figure 6. Drag "Android File Transfer" to the "Applications" folder.

Figure 6. Move Android File Transfer to Mac Applications

4. Plug in the Android device to the computer. Shown in Figure 7 is a pop-up that may appear on the Mac. If this occurs, click "OK".

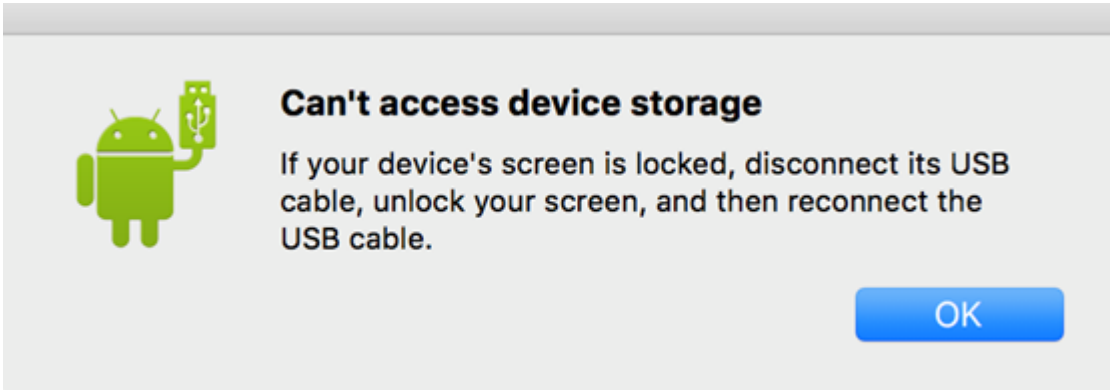


Figure 7. "Cant access device storage" Pop-up Message

5. On the Android device, a pop-up similar to the one in Figure 8 should appear. Click "ALLOW" to let the computer access the files on the device.

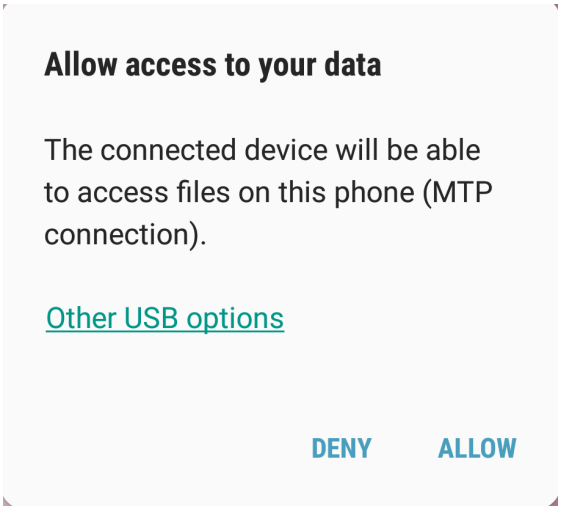


Figure 8. Pop-up Message on Android Device to Ask for Access to Data

6. On the Mac, copy IoTNode.apk into the "Download" folder. The "Download" folder is shown in Figure 9.

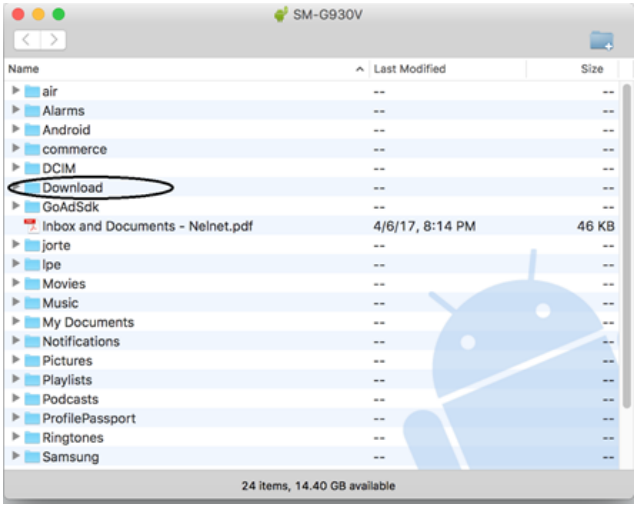


Figure 9. Android File System on Mac

3.1.2 Part 2: Install the IoTNode.apk on the Android device.

1. On the Android device, navigate to the "Download" folder.

2. In the "Download" folder, there should be the IoTNode.apk that was copied in from the computer in part 1. Double click IoTNode.apk. Figure 10 shows a security pop-up that appears when trying to install the application on the Samsung Galaxy S6 phone. Other Android device may have a similar alert message. If this pop-up occurs jump to step a. Otherwise continue on to step 3.

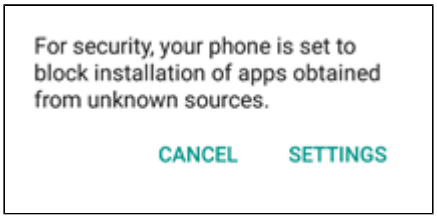


Figure 10. Android Pop-up to Alert the User that the Application Comes from an Unknown Source

a. Click "SETTINGS" on the pop-up shown in Figure 10. This should bring you to the "Security" settings menu shown in Figure 11.

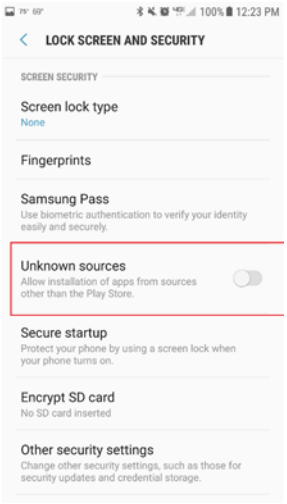


Figure 11. Android Device "Security" Settings Menu

b. Enable "Unknown sources" from the menu shown in Figure 11.

3. Click "INSTALL" from the Android application installation page shown in Figure 12.

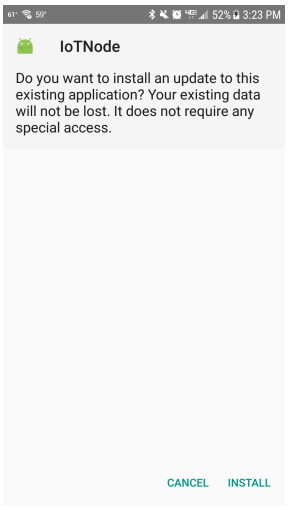


Figure 12. Android Device Application Installation Page

3.2 Installing Android Studio

To view the source code we recommend using Android Studio.

1. Download and Install Android studio from https://developer.android.com/studio/index.html. Note: We recommend you use the “Standard” Android installation.
2. Once Android studio starts up click “Open and existing Android Studio project”.
3. Open the project path "<Your Pack Path Here>\Host \android\BleApp".
4. Make sure the correct tools and packages are installed.
    a. Navigate to Tools->Android->SDK Manager to open the SDK manager.

b. On the tab “SDK Platform” make sure the SDK packages from Android 4.1 (Jelly Bean) to Android 7.1.1 (Nougat) are installed. This can be seen in Figure 13.

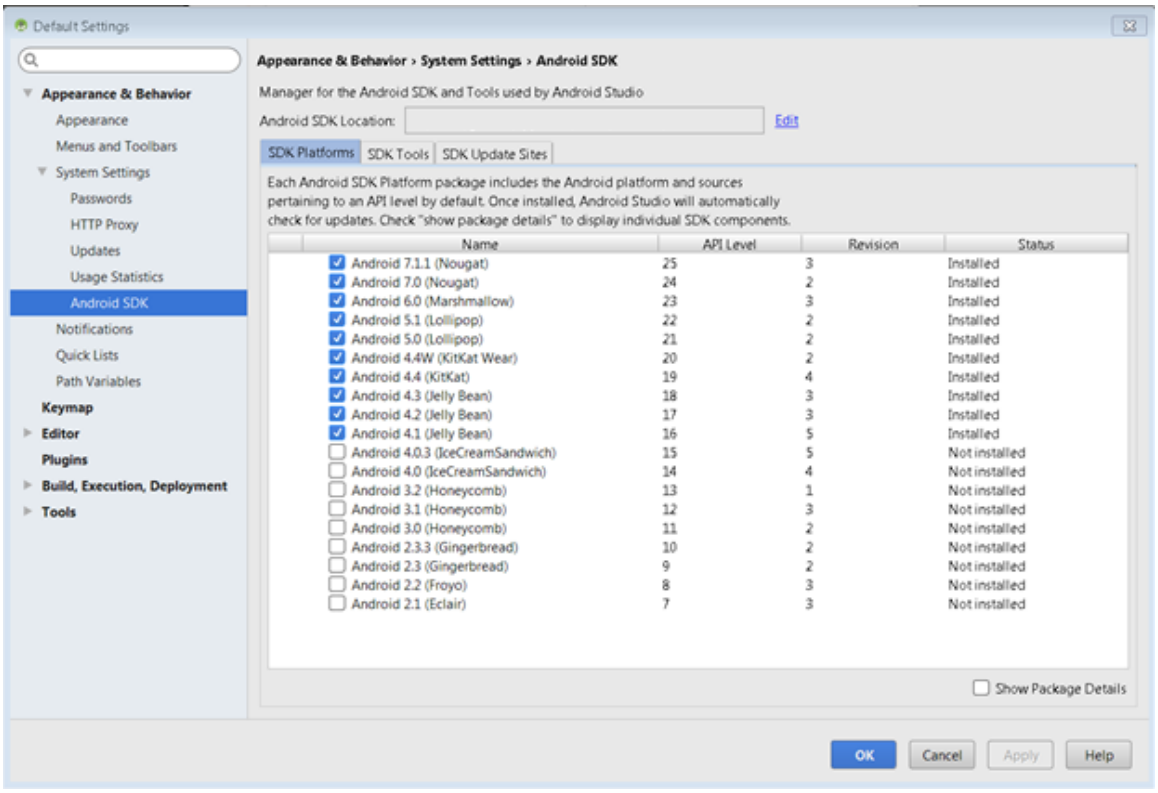


Figure 13. SDK Platform Tab on the SDK Manager

c. On the tab “SDK Tools” make sure the following tools are installed. This can be seen in Figure 14.

- Android SDK Platform-Tools
- Google Repository
- Intel x86 Emulator Accelerator (HAXM installer)
- Google USB Driver
- Documentation for Android SDK
- Android SDL Tools 25.2.5
- Android SDK Build-Tools

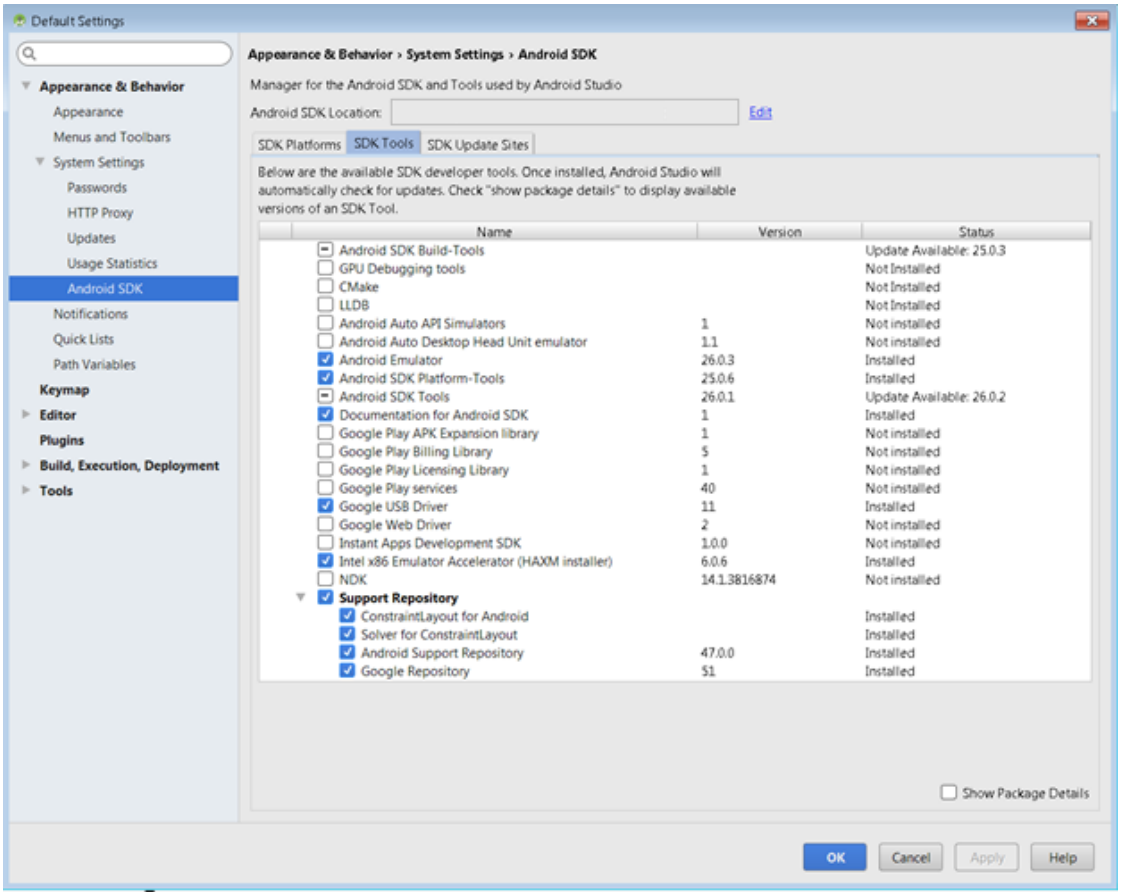


Figure 14. SDK Tools Tab on the SDK Manager

4 Using the IoTNode Application

This section of the document will discuss the current capabilities of the Android application, IoTNode, pertaining to Bluetooth. It expects the reader to have some background knowledge of Bluetooth services and profiles. For more information on the Bluetooth Specification please go to https://www.bluetooth.com/specifications.

4.1 Discovering Bluetooth Devices

When the application starts up it launches the Android Activity, ApplicationMain. Figure 14 shows the ApplicationMain layout. At a high level, this Activity enables the user to scan for remote Bluetooth devices. This can be started by pressing the button “SCAN”. This will begin the Bluetooth discovery process for 12 seconds. At any point during this, the user can press “STOP SCAN” to end the discovery process.

Figure 14. ApplicationMain Layout Before Discovery Has Taken Place

4.2 Connecting to Bluetooth Devices

Once the user has completed Bluetooth discovery, the ApplicationMain layout will reflect the list of connectable remote devices, like that in Figure 15. The user can click on a device to connect to it. The application will allow 7 seconds for the connection to take place. The user will be notified if the connection process fails. While there are many reasons a connection attempt can fail, a couple of more common reasons to look out for would be if the device has turned off, gone out of range, or if the remote device requires a certain level of security in order to establish a connection. This could mean it is looking for the user to enter a passkey or out of band data.

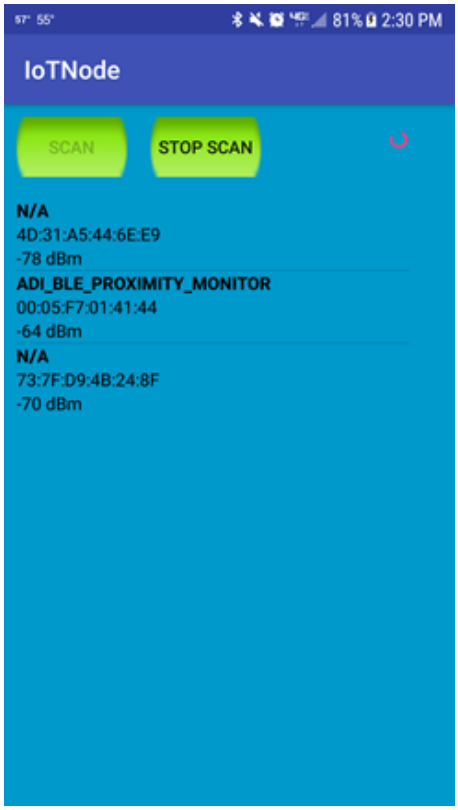


Figure 15. ApplicationMain Layout After Discovery Has Taken Place

Once a connection has been made, the application will attempt to discover the remote devices' services, shown in Figure 16. Based on the services discovered, the application will either launch the appropriate Bluetooth Profile Activity, or alert the user that the Bluetooth profile is not supported at this time. If you would like to add support for this profile, please read the section Developing the IoTNode Application.

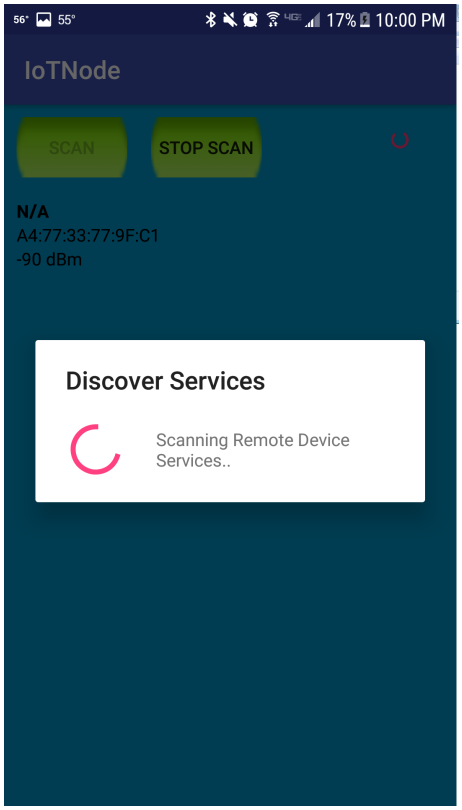


Figure 16. ApplicationMain Layout Discovering Services

4.3 Supported Bluetooth Profiles

4.3.1 Find Me Profile

The IoTNode application has the ability to act as a Find Me Locator. When the ApplicationMain Activity is scanning for services, it will check to see if the remote device has the Immediate Alert Service and 4 services in total. This combination should uniquely identify the Find Me Profile. According to the Bluetooth specifications, the Find Me Target Profile should have the Immediate Alert Service enabled. The other three Services that it should have are the Generic Access Service, Generic Attribute Service and a Custom Service with the UUID 0x7000. Once the application has identified the Find Me Profile, it will launch the FindMeLocatorActivity Activity. The layout for this Activity is shown in Figure 17.

When the “NO ALERT” button is pressed, a No Alert Level Immediate Alert will be sent to the remote device. When the “MEDIUM ALERT” button is pressed, a Medium Alert Level Immediate Alert will be sent to the remote device. When the “HIGH ALERT” button is pressed, a High Alert Level Immediate Alert will be sent to the remote device. When the “DISCONNECT” button is pressed, the application will disconnect from the remote device and the user will be brought back to the ApplicationMain Activity.

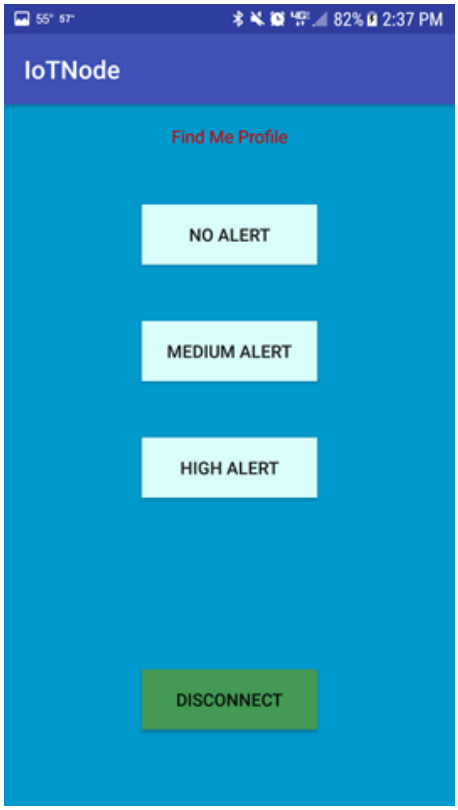


Figure 17. FindMeLocatorActivity Layout

4.3.2 Proximity Profile

The IoTNode application has the ability to act as a Proximity Monitor. When the ApplicationMain Activity is scanning for services, it will check to see if the remote device has the Link Loss Service and 5 services in total. This combination should uniquely identify the Proximity Profile. According to the Bluetooth specifications, the Proximity Monitor should have the Link Loss Service enabled. For the sake of this application, the android application also looks for the optional Immediate Alert Service. The other three services that it should have are the Generic Access Service, Generic Attribute Service and a Custom Service with the UUID 0x7000. Once the application has identified the Proximity Profile, it will launch the ProximityMonitorActivity Activity. The layout for this Activity is shown in Figure 18.

The user can move the sliding bar to adjust the path loss threshold value. If the path loss of the remote device exceeds that of the threshold, the application will send a Medium Alert Level Immediate Alert to the remote device. When the “DISCONNECT” button is pressed, the application will disconnect from the remote device and the user will be brought back to the ApplicationMain Activity.

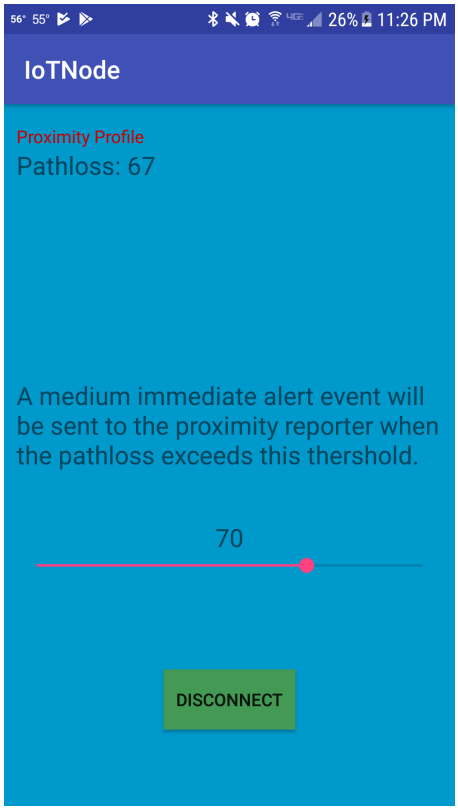


Figure 18. ProximityMonitorActivity Layout

4.3.3 Data Exchange Profile

The data exchange profile is a Custom Profile specific to the ADICUP3029 board. This profile enables the ADICUP3029 to send sensor data to the IoTNode application via Bluetooth. Currently 5 data formats are supported: accelerometer data, carbon monoxide data, visible light data, temperature data and a string. When the ApplicationMain Activity is scanning for services, it will check to see if the remote device has the Data Exchange Custom Service and 4 services in total. This combination should uniquely identify the Data Exchange Profile. The other three services that it should have are the Generic Access Service, Generic Attribute Service and a Custom Service with the UUID 0x7000. Once the application has identified the Data Exchange Profile, it will launch the SensorDisplayActivity Activity.

The layout for this Activity will vary based on the sensor(s) sending data. For every sensor sending data, a graph or string will be added to the ListView. The user can zoom in on a specific graph by clicking on the sensor name or ID. The zoomed image can be seen in Figure 19. The sensor graphs will interpret the data based on the data packet being sent from the ADICUP3029. More about the packet structure can be found in the SensorSoftware_Users_Guide.pdf. When the "DISCONNECT" button is pressed, the application will disconnect from the remote device and the user will be brought back to the ApplicationMain Activity.

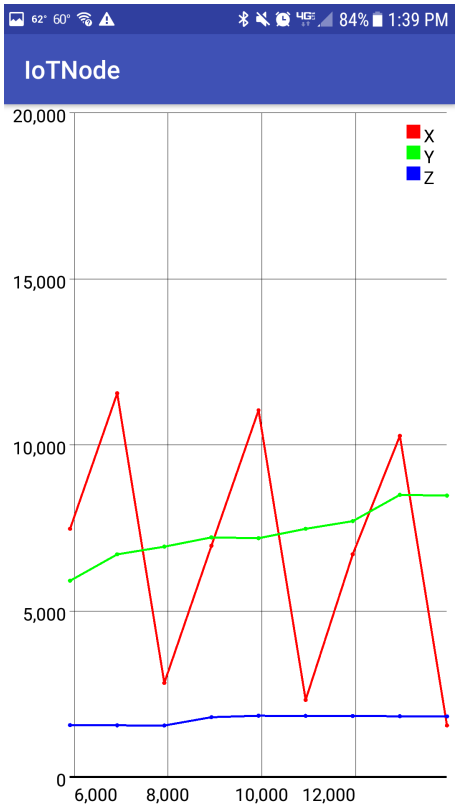


Figure 19. Zoom View of Accelerometer Data

Accelerometer

The graph of a accelerometer contains 3 data points: X, Y and Z. An example of this graph can be seen in in Figure 20. The graph will update every time new data is sent to this particular sensor with the same ID.

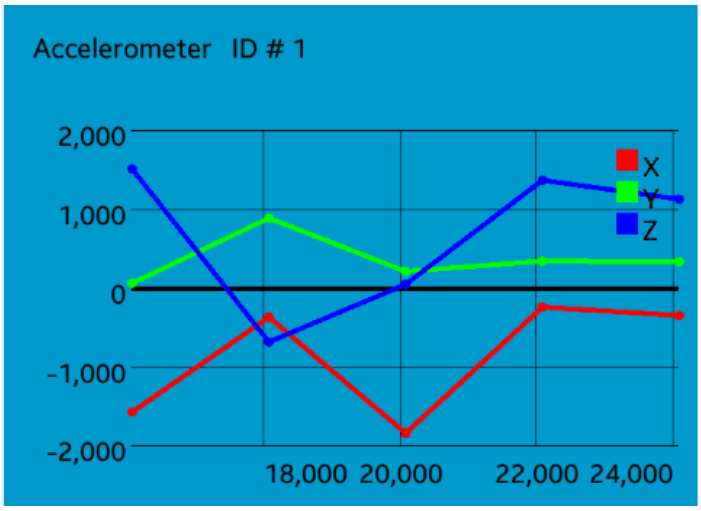


Figure 20. Accelerometer Graph in the SensorDisplayActivity ListView

Temperature Sensor

The graph of a temperature sensor contains 2 data points: Celsius and Fahrenheit. An example of this graph can be seen in in Figure 21. The graph will update every time new data is sent to this particular sensor with the same ID.

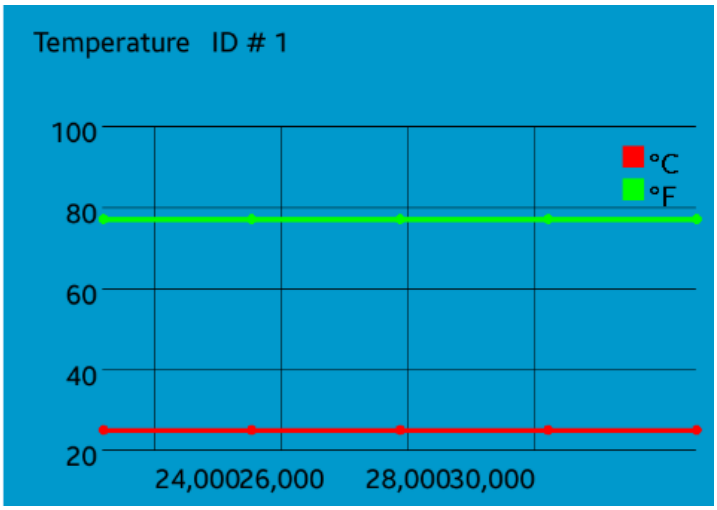


Figure 21. Temperature Graph in the SensorDisplayActivity ListView

Carbon Monoxide Sensor

The graph of a carbon monoxide sensor contains one data point: PPM, gas concentration in parts per million. An example of this graph can be seen in in Figure 22.The graph will update every time new data is sent to this particular sensor with the same ID.

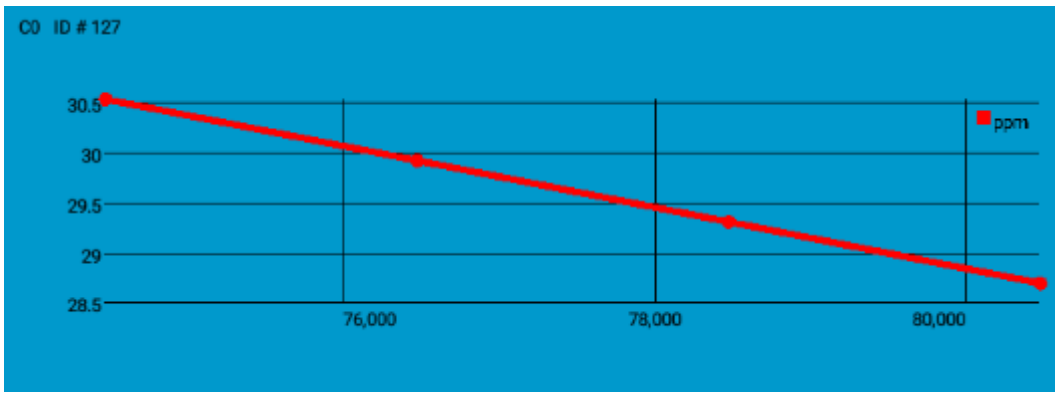


Figure 22. Carbon Monoxide Graph in the SensorDisplayActivity ListView

Visible Light Sensor

The graph of a visible light sensor contains 3 data points: red light intensity, green light intensity, and blue light intensity. An example of this graph can be seen in in Figure 23. The graph will update every time new data is sent to this particular sensor with the same ID.

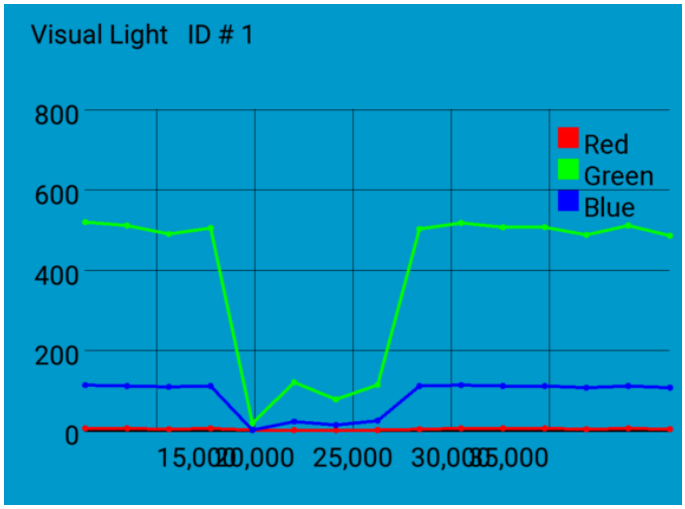


Figure 23. Visible Light Graph in the SensorDisplayActivity ListView

String Data

When the user sends string data, the string will replace where the graph would appear on that particular list view. An example of this View can be seen in in Figure 24. The string will update every time new data is sent as a string with this ID.

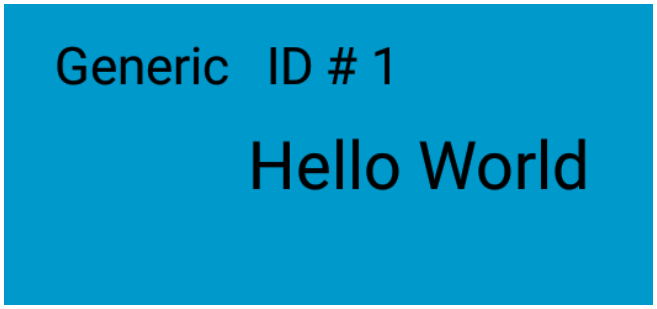


Figure 24. "Hello World" string in the SensorDisplayActivity ListView

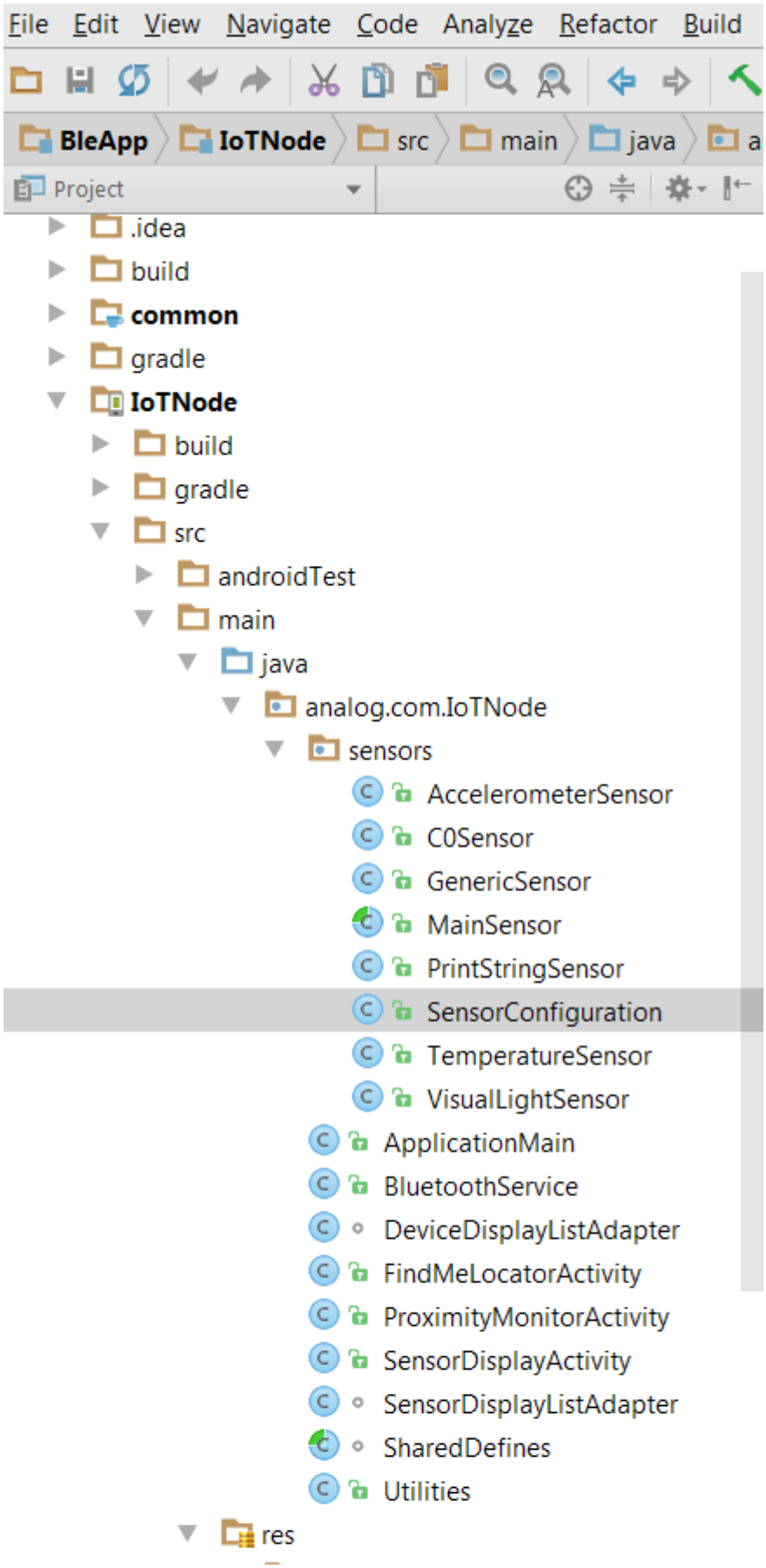
5 Developing the IoTNode Application

This section will cover the development process for the IoTNode Application. It will go over the basics of using Android Studio to modify the code to meet custom requirements on a computer running Windows OS.

5.1 Developing with Android Studio

This section assumes that Android Studio has been installed on your machine. If this is not the case, please read section Installation Components to learn about Android Studio installation.

The first thing to do is open the project using Android Studio. Once Android studio starts up click “Open an existing Android Studio project”. Open the project "<Your Pack Path Here>\Host \android\BleApp". This should bring you to a project directory that looks like the one shown in Figure 25.


File Edit View Navigate Code Analyze Refactor Build
BleApp IoTNode src main java a
Project
.idea
build
common
gradle
IoTNode
build
gradle
src
androidTest
main
java
analog.com.IoTNode
sensors
AccelerometerSensor
C0Sensor
GenericSensor
MainSensor
PrintStringSensor
SensorConfiguration
TemperatureSensor
VisualLightSensor
ApplicationMain
BluetoothService
DeviceDisplayListAdapter
FindMeLocatorActivity
ProximityMonitorActivity
SensorDisplayActivity
SensorDisplayListAdapter
SharedDefines
Utilities
res

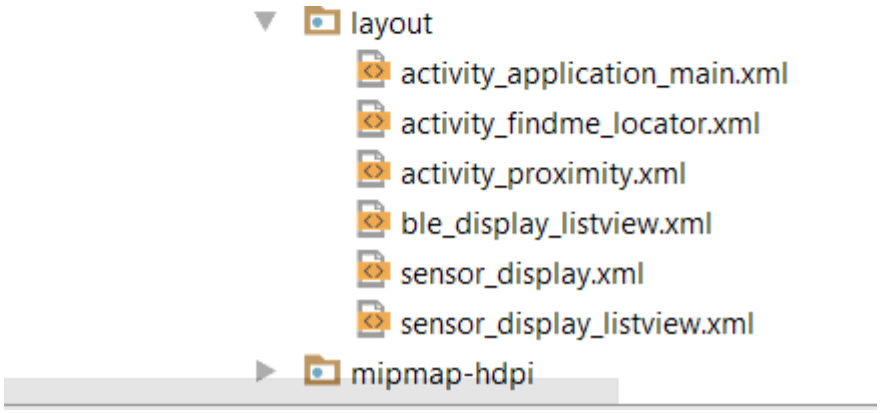


Figure 25. IoTNode Project Directory Structure

5.1.1 Installing Drivers for the Android Device

Once the project has been opened the next step is to install the drivers for the Android device that will be used during this testing process. In order for Android Studio to recognize the device for debugging, the proper drivers need to be installed. While Android Studio has the ability to run on a virtual device, that will not be covered in the scope of this manual as it is not recommended due to emulators' the lack of support for Bluetooth. Before installing the drivers, plug the Android device into your computer. Next, open "Windows Update". This can be found in the "Control Panel" menu of a machine running Windows OS. Click the "Check online for updates from Windows Update", shown in Figure 26.

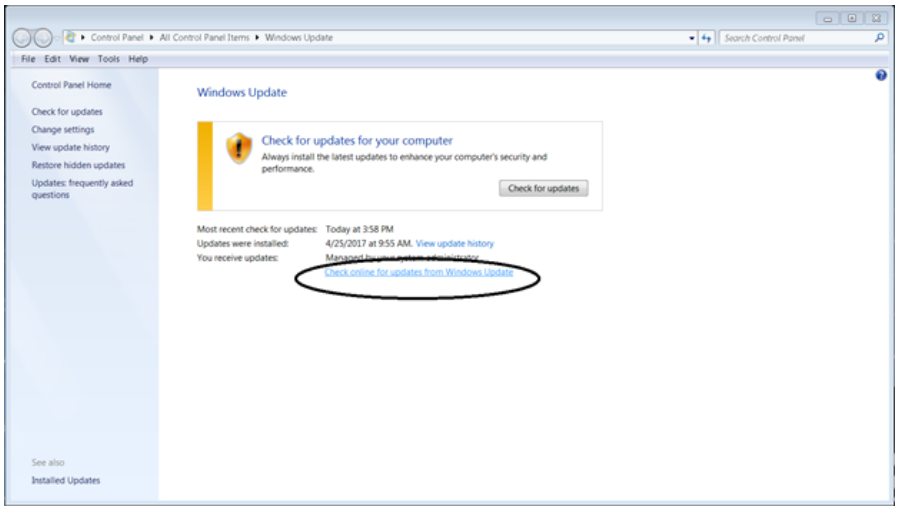


Figure 26. Windows Update Screen Before an Update is Run

The search for a windows update can vary in time, so please be patient. Once the search has completed there should a notification similar to that in Figure 27. Click "1 important update is available" highlighted in blue.

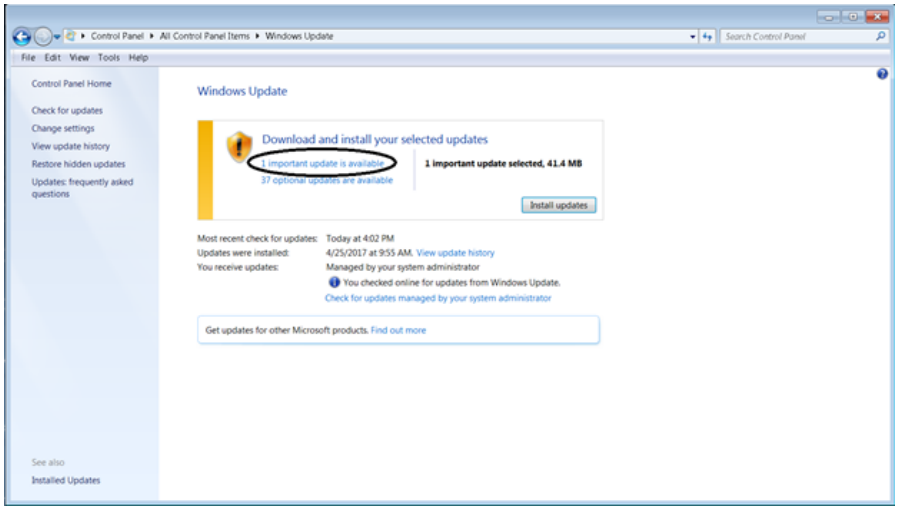


Figure 27. Windows Update Screen After an Update has Completed

This will bring you to a page to view the updates found. There should be an update for the Android device that is currently plugged into the computer.

Note: If that is not the case then this is either not a supported Android device, or the driver is already installed successfully.

Make sure this box is checked like that shown in Figure 28 and click "OK" to go back to the prior screen. Please note that the only box that needs to be checked is the one for the driver. We do not ask that you install any other updates. Please use your discretion when deciding which boxes to leave checked. Then click "Install Updates".

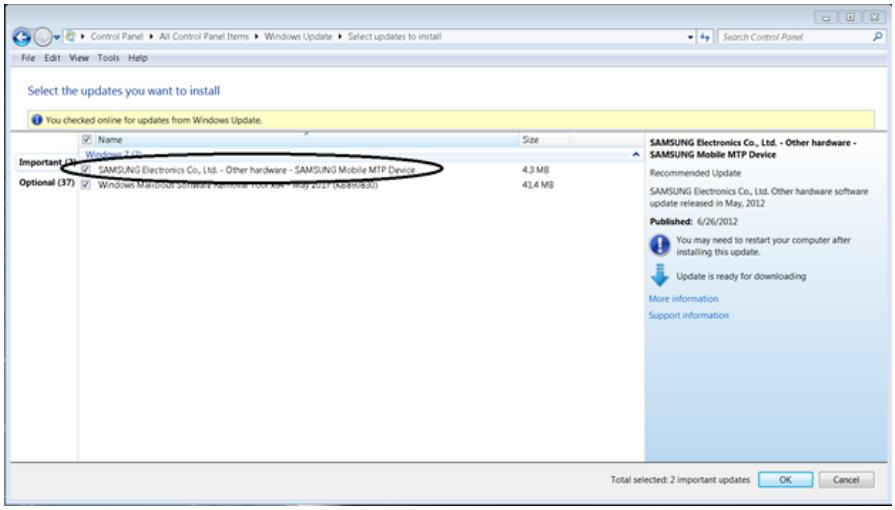


Figure 28. Windows Update Screen Found Samsung Android Driver

To confirm that the driver installation was successful, open device manager and see if the Android Device has been found, shown in figure 29.

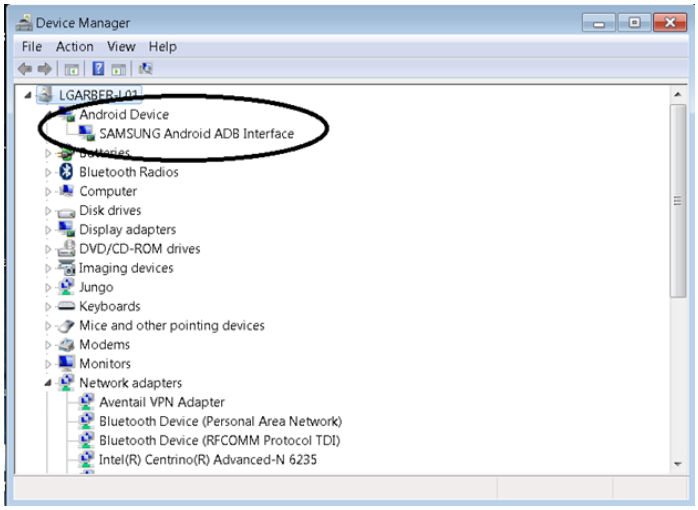


Figure 29. Device Manager with Android Device

5.1.2 Debugging in Android Studio

Once the drivers have been installed you can begin debugging the application. Make sure your Android device is plugged in during this process. To begin, go to “Run”->”Debug 'IoTNode'”. Once you click on this, a menu should pop up that looks like the one in Figure 30. Select your physical Android Device and click “OK”. This will download the IoTNode application to the device.

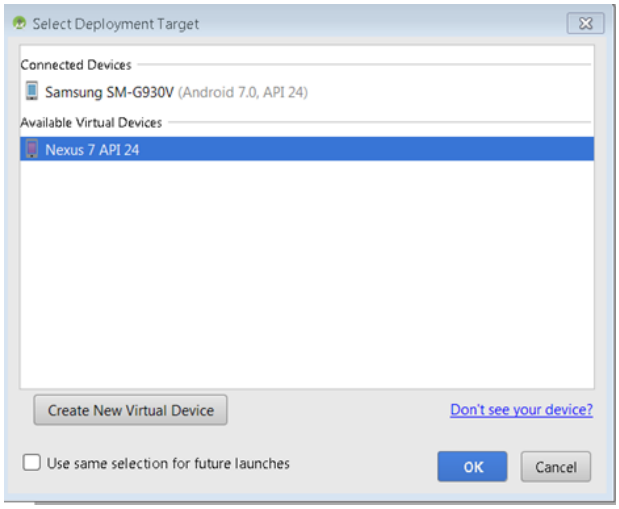


Figure 30. Android Studio Select Deployment Target Menu

The tool that we recommend for debugging this application is called the "Android Monitor", shown in Figure 31. This is useful in debugging memory leaks as well as rogue Activities and Services. It shows the console message in real time as well as the memory, CPU, network and GPU usage. This can be opened by clicking View->Tools Windows->Android Monitor.

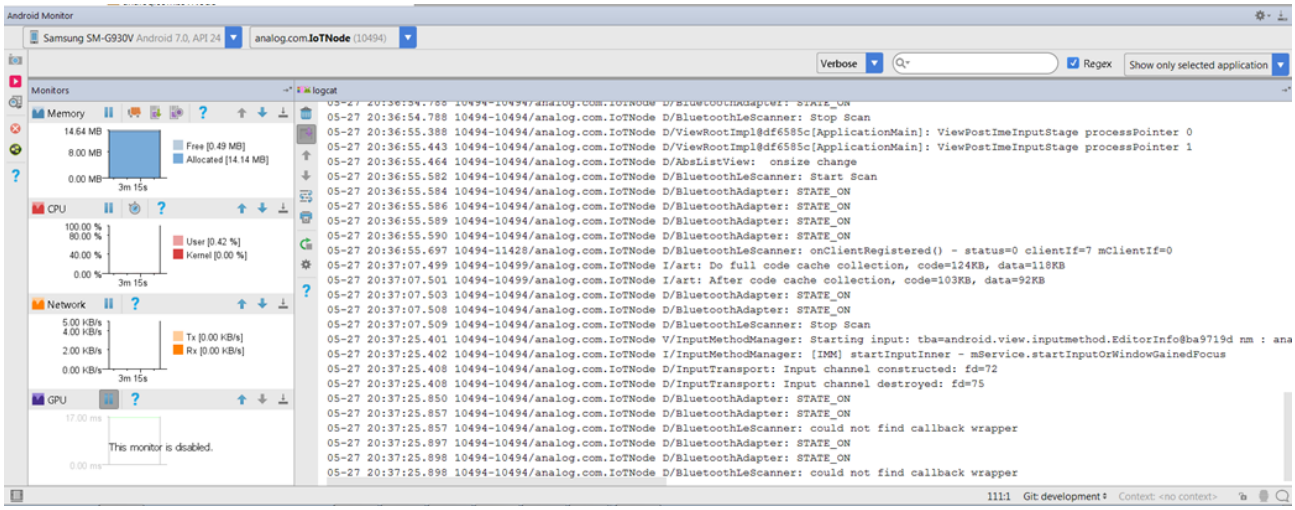


Figure 31. Android Monitor

5.2 Overview of the IoTNode Application Software

This section will cover at a very high level the basic Bluetooth data flow of the IoTNode application. The overall data flow can be seen in Figure 32.

The IoTNode application starts on the ApplicationMain. This is a UI driven activity. The user has the option to start discovery, stop discovery and connect to a remote device. When the user chooses any of these options, a message will be sent to the BluetoothService via the BroadcastReceiver. If the user has chosen to connect to a remote device, ApplicationMain will start the Activity that corresponds with the Bluetooth Services discovered on that device.

The BluetoothService is the only part of the application that uses the Bluetooth object. Anything that needs to be done related to Bluetooth passes through this Service. Once the Bluetooth action has completed, the BluetoothService will use its BroadcastReceiver to send a message back to the ApplicationMain with the result of the Bluetooth action performed.

There are 3 different profile Activites: FindMeLocatorActivity, ProximityMonitorActivity, and the SensorDisplayActivity. Each of these correspond to a unique set of services also known as a Bluetooth profile. Once one of these services has been started it will communicate with the BluetoothService to send and receive data to the remote device via Bluetooth Characteristics.

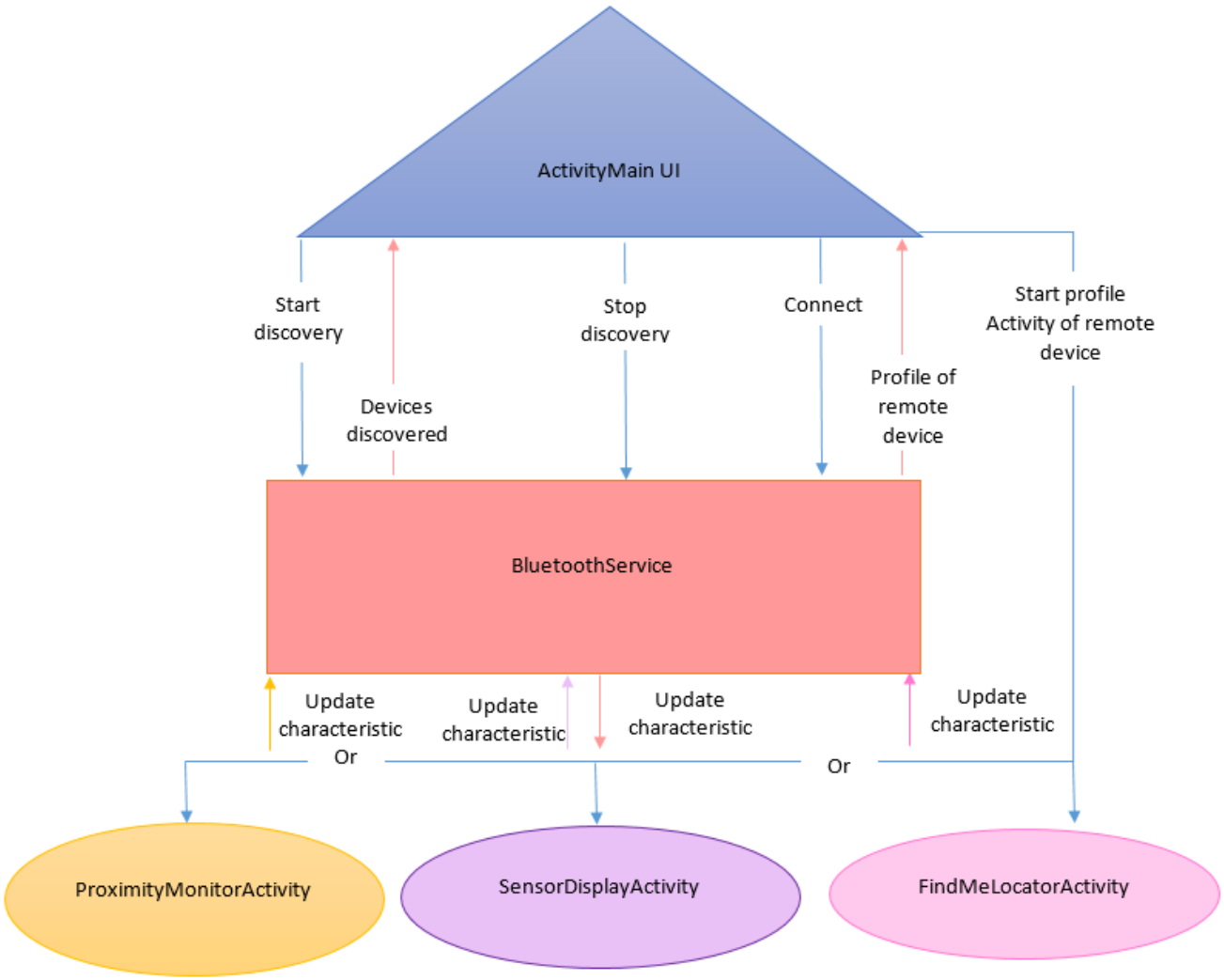


Figure 32. IoTNode Data Flow Diagram

5.3 How to add a New Sensor

1. Create the Java Class. Right click on the "sensors" folder in directory structure in Android Studio. Click NewJava Class.
2. Name the Java class based on the name of the sensor and have it extend the MainSensor. Implement all of the methods you want it to have.
    a. ParseData(): Begin by extracting your sensor data from the byte[] sensorData object. Once extracted, you can manipulate it in any way you like. After you have all the points for the one graph of this sensor, organize them into a double[] array and pass it off in the return statement. Make sure both the double[] and legend[] match in size.

3. Add the value for this new sensor type to the file "analog.com.IotNode.SharedDefines". In this file there will be definitions with the suffix "_SENSOR", shows in Figure 33. This int value is what the sensor would be sending in the field "ID of Sensor". See SensorSoftware_Users_Guide.pdf for more details on the packet format.

SharedDefines.java ×

SharedDefines | PROXIMITY_SHORT_UUID

```
66      /**
67       * Generic Sensor ID type. This sensor will be whatever the registraion packet specifies.
68       * Currently not supported. Note: If you change these be  sure to update their corresponding
69       * values for the ADICUP3029 in adi_packet.h
70       */
71      static final int GENERIC_SENSOR = 0;
72      /**
73       * Accelerometer 2G Sensor ID type. Note: If you change these be sure to update their
74       * corresponding values for the ADICUP3029 in adi_packet.h
75       */
76      static final int ACCELEROMETER_2G_SENSOR = 1;
77      /**
78       * CO Sensor ID type. Note: If you change these be sure to update their corresponding
79       * values for the ADICUP3029 in adi_packet.h
80       */
81      static final int CO_SENSOR = 2;
82      /**
83       * Temperature Sensor ID type. Note: If you change these be sure to update their corresponding
84       * values for the ADICUP3029 in adi_packet.h
85       */
86      static final int TEMPERATURE_SENSOR = 3;
87      /**
88       * Visual Light Sensor ID type. Note: If you change these be sure to update their corresponding
89       * values for the ADICUP3029 in adi_packet.h
90       */
91      static final int VISUAL_LIGHT_SENSOR = 4;
92      /**
93       * Print String Sensor ID type. This is just a string sent from the application. Note: If you
94       * change these be sure to update their corresponding values for the ADICUP3029 in adi_packet.h
95       */
```

Figure 32. ID of Sensor Definitions

4. In analog.com.IotNode.SensorDisplayAdapter locate the switch statement in the method handlePacket(). In this switch statement, add your new "XXX_SENSOR" value to the list and have it create a new instance of your sensor class. This switch statement can be seen in Figure 33.

```
/* If the sensor does not already exist in the list, add it based on the sensor type. This
 * is important because each sensor type is displayed differently.
 */
switch (sensorType) {
    case SharedDefines.VISUAL_LIGHT_SENSOR:
        mSensorList.add(new VisualLightSensor(sensorId, timeStamp, data));
        break;
    case SharedDefines.ACCELEROMETER_2G_SENSOR:
        mSensorList.add(new AccelerometerSensor(sensorId, timeStamp, data));
        break;
    case SharedDefines.ACCELEROMETER_4G_SENSOR:
        mSensorList.add(new AccelerometerSensor(sensorId, timeStamp, data));
        break;
    case SharedDefines.ACCELEROMETER_8G_SENSOR:
        mSensorList.add(new AccelerometerSensor(sensorId, timeStamp, data));
        break;
    case SharedDefines.TEMPERATURE_SENSOR:
        mSensorList.add(new TemperatureSensor(sensorId, timeStamp, data));
        break;
    case SharedDefines.CO_SENSOR:
        mSensorList.add(new C0Sensor(sensorId, timeStamp, data));
        break;
    case SharedDefines.GENERIC_SENSOR:
        break;
    case SharedDefines.PRINT_STRING_SENSOR:
        mSensorList.add(new GenericSensor(sensorId, data));
        break;
}
```

Figure 33. Switch Statement to Create New Instance of Sensor