**Android-Arduino CAN Project**

The purpose of this project was to demonstrate communication between a car and an Android tablet, using an Arduino as a bridge.

The Android tablet was connected to the Arduino over Bluetooth, and the Arduino was connected to the car’s OBD-II port, using an OBD-II to UART board as an interface. An app on the Android tablet displayed readings from the car on a virtual dashboard.

**Communication**

To communicate over Bluetooth, the standard Android and Arduino APIs were used. They provide simple libraries to use on both the Android and Arduino side to allow for communication.

**Android**

Several pieces are needed to establish Bluetooth communication on the Android side. Firstly, you must pair the android device to the Bluetooth device. After the devices are paired, Everything else is handled internally. The communication is handled through a standard Java InputStream and OutputStream. Buffers of bytes can be written, serially, to the OutputStream, or read from the InputStream, which forms the basis of communication between the Android device and the Arduino board.

The basic components of establishing a connection are a BluetoothAdapter, BluetoothSocket, and BluetoothDevice. The BluetoothAdapter and BluetoothDevice classes are used to find and establish a Bluetooth socket connection with the device.

Input is handled by a LinkedBlockingQueue, which is a thread safe derivation of the Queue class. As the android device gets messages from the Arduino, it stores them in this queue so that they can be retrieved at any time. The majority of the input handling is done in the following code snippet.

**for**(**int** i = 0; i < msgLen; i++) {

**try** {

inputBuffer.put(buffer[i]);

**if**((**char**)buffer[i] == 'N')

foundNullTerminatorFlag = **true**;

**if**(foundNullTerminatorFlag && buffer[i] == 0)

storeMsg();

} **catch** (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

Output is handled by writing bytes to the OutputStream, which can be seen in the following method.

**public void** sendByte(**byte** value) {

**byte** buffer[] = new **byte**[1];

buffer[0] = value;

**if**(outputStream != **null**) {

**try** {

outputStream.write(buffer);

} **catch** (IOException e) {

log("Send failed: " + e.getMessage());

}

} **else** {

log("Send failed: outputStream was null");

}

}

**Arduino**

The interface with the Bluetooth hardware is simple, as it’s just a serial port. We created a Bluetooth library, for use with Arduino programs, to make using Bluetooth simpler. The library handles initializing the proper I/O pins, powering up the Bluetooth device, and issuing connection status updates coming from the device.

To begin communication over Bluetooth, the device first needs to be initialized. Then, it can be used as a standard serial port. We chose to use a SoftwareSerial port, however, in order to use non-standard input and output pins for the serial port. This provides more flexibility with how the hardware is configured.

To initialize the Bluetooth device, a Bluetooth object is constructed, then Bluetooth.beginBluetooth() is called. beginBluetooth() handles all of the commands to initialize the device, and will return false if the initialization fails.

To communicate, bytes can be sent across Bluetooth using Bluetooth.sendByte() or Bluetooth.sendStringRaw(). To receive, Bluetooth.bytesAvailable() must be called to check how many bytes are available for reading. Then, each byte can be read individually with Bluetooth.readByte().Bluetooth.process() must be called at the top of loop().

To check the status of the connection, Bluetooth.isConnected() is used. This is a simple example program to show how the library can be used. 11 and 3 are the RX and TX pins, then the name of the Bluetooth device, then whether or not the Bluetooth library should send log messages on Serial, and lastly, how many times it should retry initialization.

SoftwareSerial bluetoothSerial(11, 3);

**Bluetooth** bluetooth(“AndroidArduinoBluetooth”, bluetoothSerial, true, 3);

**void** **setup**() {

**if**(bluetooth.beginBluetooth()) {

bluetooth.sendStringRaw(“Connected!”);

}

}

**void** **loop**() {

bluetooth.process();

**if**(bluetooth.isConnected()) {

**while**(bluetooth.bytesAvailable()) {

**Serial**.**write**(bluetooth.readByte());

}

}

}

**Android App**

The Android app consists mainly of a tachometer, a speedometer, a connect/disconnect button, and a panel of several other gauges. It also has the ability to track a driving session, reporting the average speed and total distance traveled. The connect/disconnect button allows for manual control over the Bluetooth connection, although the app does try to automatically connect and disconnect when opened and closed.

The Arduino gathers readings from the car as fast as the ODB-II to UART hardware will allow, then sends all of the raw data to the Android tablet. The tablet then parses the data and calculates values for each parameter, then updates the on-screen gauges.

**Communication**

To communicate over Bluetooth, the standard Android and Arduino APIs were used. They provide simple libraries to use on both the Android and Arduino side to allow for communication.

**Android**

Several pieces are needed to establish Bluetooth communication on the Android side. Firstly, you must pair the android device to the Bluetooth device. After the devices are paired, Everything else is handled internally. The communication is handled through a standard Java InputStream and OutputStream. Buffers of bytes can be written, serially, to the OutputStream, or read from the InputStream, which forms the basis of communication between the Android device and the Arduino board.

The basic components of establishing a connection are a BluetoothAdapter, BluetoothSocket, and BluetoothDevice. The BluetoothAdapter and BluetoothDevice classes are used to find and establish a Bluetooth socket connection with the device.

Input is handled by a LinkedBlockingQueue, which is a thread safe derivation of the Queue class. As the android device gets messages from the Arduino, it stores them in this queue so that they can be retrieved at any time. The majority of the input handling is done in the following code snippet.

**for**(**int** i = 0; i < msgLen; i++) {

**try** {

inputBuffer.put(buffer[i]);

**if**((**char**)buffer[i] == 'N')

foundNullTerminatorFlag = **true**;

**if**(foundNullTerminatorFlag && buffer[i] == 0)

storeMsg();

} **catch** (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

Output is handled by writing bytes to the OutputStream, which can be seen in the following method.

**public void** sendByte(**byte** value) {

**byte** buffer[] = new **byte**[1];

buffer[0] = value;

**if**(outputStream != **null**) {

**try** {

outputStream.write(buffer);

} **catch** (IOException e) {

log("Send failed: " + e.getMessage());

}

} **else** {

log("Send failed: outputStream was null");

}

}

**Arduino**

The interface with the Bluetooth hardware is simple, as it’s just a serial port. We created a Bluetooth library, for use with Arduino programs, to make using Bluetooth simpler. The library handles initializing the proper I/O pins, powering up the Bluetooth device, and issuing connection status updates coming from the device.

To begin communication over Bluetooth, the device first needs to be initialized. Then, it can be used as a standard serial port. We chose to use a SoftwareSerial port, however, in order to use non-standard input and output pins for the serial port. This provides more flexibility with how the hardware is configured.

To initialize the Bluetooth device, a Bluetooth object is constructed, then Bluetooth.beginBluetooth() is called. beginBluetooth() handles all of the commands to initialize the device, and will return false if the initialization fails.

To communicate, bytes can be sent across Bluetooth using Bluetooth.sendByte() or Bluetooth.sendStringRaw(). To receive, Bluetooth.bytesAvailable() must be called to check how many bytes are available for reading. Then, each byte can be read individually with Bluetooth.readByte().Bluetooth.process() must be called at the top of loop().

To check the status of the connection, Bluetooth.isConnected() is used. This is a simple example program to show how the library can be used. 11 and 3 are the RX and TX pins, then the name of the Bluetooth device, then whether or not the Bluetooth library should send log messages on Serial.

**Bluetooth** bluetooth(11, 3, “AndroidArduinoBluetooth”, true);

**void** **setup**() {

**if**(bluetooth.beginBluetooth()) {

bluetooth.sendStringRaw(“Connected!”);

}

}

**void** **loop**() {

bluetooth.process();

**if**(bluetooth.isConnected()) {

**while**(bluetooth.bytesAvailable()) {

**Serial**.**write**(bluetooth.readByte());

}

}

}

**Arduino Firmware**

The Arduino firmware isn’t too complex. When it starts up, it initializes the Bluetooth, then initializes the OBD-II to UART board. The OBD-II board automatically detects different OBD-II standards, and provides a simple command-line interface over a serial connection. It’s meant to be used with a computer, but an Arduino can handle the communication.

Once everything is ready and the Android tablet has been connected, the Arduino will start sending commands to the OBD-II board to request data from the car. For each command sent, it reads the response line by line until it sees the prompt, “>”. The response, which comes back in the form of hexadecimal bytes encoded as ASCII strings, is then chopped up to get the relevant information, then sent over Bluetooth to the Android tablet.

The performance of the Arduino side could be better. Unfortunately, the OBD-II board takes time to receive and send commands, then there’s a delay waiting for the car to respond. We’ve offloaded as much calculation as we can to the Android tablet so there’s as little delay as possible with the Arduino, but the refresh rate still tends to stay around 4Hz.

**More Information**

GitHub page for this project, which includes documentation on the libraries created:

<https://github.com/riis/AndroidArduino>

Video of the project in action:

<https://riisllc.basecamphq.com/projects/9867332/file/131143337/CAN%20app%20Video.mov>

Arduino:

<http://arduino.cc/>

Bluetooth Shield for Arduino:

<http://www.seeedstudio.com/wiki/index.php?title=Bluetooth_Shield>

OBD-II to UART board:

<https://www.sparkfun.com/products/9555>