

Creating an Bluetooth Interface for connecting an Android Device to an RFduino Microcontroller

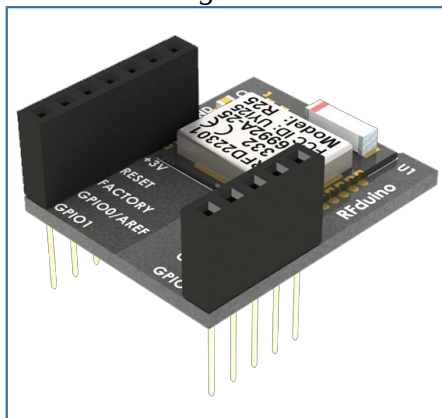
By: Parker Mojsiejenko

April 2nd, 2015

Abstract: This application note provides the necessary instruction needed to connect an Android device to an RFduino microcontroller (RFD22102). The RFduino microcontroller adds Bluetooth functionality to an easy to use, prototype friendly Arduino compatible device. Once some of the Android methods are explained, the process becomes simple, assuming programming knowledge is already known. With Bluetooth functionality, engineers and hobbyists can remotely control their projects from a distance.

Introduction

The RFduino is a microcontroller which uses hardware compatible with the Arduino programming language and utilizes a Bluetooth transmitter to communicate to other devices. The RFduino comes



with many modules to 'stack' on the microcontroller thus extending its function beyond Bluetooth integration. Figure 1 shows the RFD22102 RFduino module that is needed for all applications. A USB module is also required to program the device using the Arduino interactive development environment or from hereon referred to as an IDE. In addition to this, an IDE for the Android device is needed as well. Google provides their own IDE for programming Android devices called "Android Studio" and all of the necessary documentation for programming is available on their webpage,

<http://developer.android.com/>. Android uses the Java programming language with modified libraries to allow coding on Android devices. The Arduino IDE can be obtained through <http://www.arduino.cc/> and is a requirement for the RFduino to be programmed. Arduino uses modified C/C++ languages to program their devices.

Objective

This application note serves to provide support in connecting an Android device to the RFduino device. Both devices will be discussed to develop some familiarity and an example code will be included. The RFduino will serve as the client device in the Bluetooth connection and therefore will contain simpler

codes as it must only accept Bluetooth requests and not initiate them. Prior knowledge is assumed in development with Android and basic syntax of coding languages.

Android

Android was released in 2008 as a mobile operating system by Google Inc. It has since grown to be a standard in the mobile phone industry and according to Googles developer webpage, it currently powers hundreds of millions of devices in 190 countries across the world. Due to its open source code and absence of required fees to develop for, Android is one of the most widely used mobile platforms in the world and is a great system to implement into projects and designs. Android uses Java as its programming language and shares the same capabilities as stand-alone Java applications.

As this is not an application note on Java itself and instead the use of an Android device to connect to another Bluetooth device, the RFduino in this case, there will be an assumption that knowledge of the Java language is already known. Google has included Bluetooth functionality in their Bluetooth API and it is capable of the following key features:

- Scan for other Bluetooth devices
- Query the local Bluetooth adapter for paired Bluetooth devices
- Establish RFCOMM channels
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections

Android uses the **BluetoothAdapter** class to perform most of the basic Bluetooth functions, such as scanning for devices, creating the remote Bluetooth device object, querying paired devices and listening for communications to other devices. It is the entry-point for Bluetooth in the Android system. This class will also create a **BluetoothServerSocket** object for communications with other devices. This class returns a **BluetoothSocket** object upon a successful connection. An example of this is shown in code below which was taken from Googles Bluetooth API documentation.

```
BluetoothServerSocket tmp = null;
try {
    // MY_UUID is the app's UUID string, also used by the client code
    tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
} catch (IOException e) { }
mmServerSocket = tmp;
```

```
BluetoothSocket socket = null;
// Keep listening until exception occurs or a socket is returned
while (true) {
    try {
        socket = mmServerSocket.accept();
    } catch (IOException e) {
        break;
    }
    // If a connection was accepted
    if (socket != null) {
        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(socket);
        mmServerSocket.close();
        break;
    }
}
```

The code sets up the **BluetoothServerSocket** with the name of its service and its UUID, or Universally Unique Identifier, which as expected gives the service a unique identifier. The device will then continuously accept the connection even if it doesn't exist yet. When the **BluetoothSocket** is created it will be sent to the **manageConnectedSocket()** method and the **ServerSocket** is closed to free up resources. This is the basic method of establishing a connection to another Bluetooth enabled device from Android. Utilizing the connection for data transfer can be more difficult but establishing a connection is relatively straightforward. The above code is the bare minimum to implement into an app.

RFduino

The RFduino uses a 16MHz ARM Cortex-M0 CPU and runs off 3V of power. It is a powerful microcontroller and is valuable in prototyping designs. There is also a power module available that uses AAA batteries to provide power if you choose to power the microcontroller independently. The device is programmed using the Arduino language which is a modified C/C++ variant called Wiring. The main differences are the use of the **setup()** and **loop()** functions in the code operation. The **setup()** function is called when the Arduino, or in this case the RFduino, is powered up and is run only once.

The **loop()** function is then run continuously until the device is powered down. All codes will include those two functions.

The RFduino requires a library for the bluetooth capabilities to be included in the code and is used in all programs run on the RFduino. On the next page there is an included example 'sketch' (Arduino code) of a program for an RFduino. It is a simple program that does the following: While advertising, the green LED will be lit and when connected to a device the red LED will be lit. The LED pins are set in the setup function and the Bluetooth is started with the **RFduinoBLE.begin()** call. This starts the Bluetooth service in the background. The service calls various functions when events are triggered, such as connecting and disconnecting from a device. The code below turns on and off the red LED based on what function is called. Also, a function is called when the device starts or stops advertising that controls the green LED. The loop function keeps the device in low power mode to save power while using the Bluetooth function.

The RFduino acts as the client device in Bluetooth connection. This allows a simple setup on the microcontroller which is less documented than other Bluetooth capable devices like the Android system.

Sample Arduino Code

```
#include <RFduinoBLE.h>

// pin 3 on the RGB shield is the green led
int advertisement_led = 3;

// pin 2 on the RGB shield is the red led
int connection_led = 2;

void setup() {
    // led used to indicate that the RFduino is advertising
    pinMode(advertisement_led, OUTPUT);

    // led used to indicate that the RFduino is connected
    pinMode(connection_led, OUTPUT);

    // start the BLE stack
    RFduinoBLE.begin();
}

void loop() {
    // switch to lower power mode
    RFduino_ULPDelay(INFINITE);
}
```

```

void RFduinoBLE_onAdvertisement(bool start)
{
    // turn the green led on if we start advertisement, and turn it
    // off if we stop advertisement

    if (start)
        digitalWrite(advertisement_led, HIGH);
    else
        digitalWrite(advertisement_led, LOW);
}

void RFduinoBLE_onConnect()
{
    digitalWrite(connection_led, HIGH);
}

void RFduinoBLE_onDisconnect()
{
    digitalWrite(connection_led, LOW);
}

```

Common Issues

Although the above examples show how to create a connection on an Android device and how to program the RFduino to accept that connection request, there still are many things that must be setup and added in order to have a successful connection. Every Android application must declare what permissions are needed to ensure a secure system. Bluetooth capabilities must be requested in the manifest file that is included in every Android app. If this is not declared with the line below, the app will fail to establish a connection through bluetooth.

`<uses-permission android:name="android.permission.BLUETOOTH" />`

It is also important to ensure that all requests made in the Android code are surrounded with 'try/catch' blocks. The way that the Java languages handles errors is one of its many positives and if the errors are not handled correctly, which in this case means 'catching' the errors properly, the code will crash. As Java is not the focus of this application note, there is no need to go any further with explaining this feature of the language. However, it is important to remember to use 'try/catch' when setting up a Bluetooth connection.

Conclusion

On the Android side, a **BluetoothAdapter** object must be created and will handle the majority of all Bluetooth functions on the Android platform. The **BluetoothServerSocket** object must also be created. This is responsible for listening to incoming requests from other Bluetooth devices. When a connection request is accepted, the **BluetoothServerSocket** will then create a **BluetoothSocket** which

will allow data transfer to occur. The final object that is worth mentioning is the **BluetoothDevice** object which serves to represent the remote device when requesting a connection or just simply querying for the remote devices information. The RFduino device must include the **#include <RfduinoBLE.h>** command in order to make use of the RFduino. This header file allows for all of the Bluetooth functions to be called on the device. In fact, it allows the capability to enable Bluetooth through the **RFduinoBLE.begin()** command. Its API can be referenced on the RFduino webpage (<http://www.rfduino.com/>) for all of the functions included with the device. Google also provides an extremely detailed API on its Bluetooth methods which can be found on their developer page (<http://developer.android.com/>). Once the connection is established you will be able to utilize all that Bluetooth has to offer and remotely control any project.

As explained, the process of creating a connection over Bluetooth between two devices is not as difficult as it may appear. Following the programming language syntax, ensuring the code is bug free, and making sure to code with good coding practices are the most important aspects of this process.

References

All information used to create this application note was taken from the following websites:

- <http://developer.android.com/>
- <http://www.rfduino.com/>
- <http://www.arduino.cc/>

More specifically Googles Bluetooth API, the RFduino API and the Arduino programming reference can be found at:

- <http://developer.android.com/guide/topics/connectivity/bluetooth.html>
- http://www.rfduino.com/wp-content/uploads/2014/03/rfduino.ble_.programming.reference.pdf
- <http://arduino.cc/en/Reference/HomePage>