

485 Android Bluetooth Sample Application

The 485 Android Bluetooth Sample Application is intended for developers attempting to communicate with a Microcom Model 485 Printer using an Android device over a Bluetooth connection. The code within this sample application will act as a reference on best practice for successful Android device/Microcom Printer Bluetooth connection and communication. The code can be copied and modified for use in the user's own application.

Requirements

1. Windows Computer
2. Android Studio 3.0 or Later
3. Android SDK Version 24
4. Microcom Model 485 Printer

Guide

Connecting

Step 1. Instantiate Bluetooth.

Step 1.1. Run the app. This creates a Bluetooth object and initializes it using the following line:

```
bluetooth = new Bluetooth(this);
```

Step 1.2. It also sets the bluetooth adapter and registers the bluetooth receiver within the Bluetooth object by calling the following line:

```
bluetooth.onStart();
```

Step 1.3. Next, it sets the Bluetooth object's callbacks so that they can handle any asynchronous Bluetooth related events. This is done in the following lines:

```
bluetooth.setCallbackOnUI(this);  
bluetooth.setDeviceCallback(new DeviceCallback() {...});  
bluetooth.setBluetoothCallback(new BluetoothCallback() {...});  
bluetooth.setDiscoveryCallback(new DiscoveryCallback() {...});
```

The code inside these callback classes defines how each event is handled. Some of these will be important later.

Step 2. Now the Bluetooth object is instantiated and ready to be interacted with. Select the Enable Bluetooth button and ensure its success. This button calls the following line:

```
bluetooth.enable();
```

to attempt to enable the Bluetooth and then uses the following function:

```
bluetooth.isEnabled();
```

to determine if the enabling succeeded or not.

Step 3. Scan for/discover devices. This can be done by selecting the Start Scanning button. Inside the onDeviceFound function of the DiscoveryCallback object, discovered devices will be passed in as the device parameter. In this example, all devices are collected in a global list. This is done using the following line (inside the onDeviceFound function):

```
discoveredDevices.add(device);
```

Wait for the status box at the bottom to display that scanning has finished along with the list of discovered devices.

Step 4. Pair with the desired device. This can be done by finding the desired device name in the list of discovered devices and copying it into the text box next to the Pair With Device Named button. If the desired device name does not appear in the list, then it was not discovered and will not be able to be paired with. Once the button is selected, the pairing will begin. This is done in the code by searching through the list of discovered devices and finding the device with a name matching the desired name entered. Once the desired device is found, pairing to the device is attempted. This is done with the following lines:

```
BluetoothDevice di = null;
boolean deviceFound = false;
for (int i = 0; i < discoveredDevices.size(); i++) {
    di = discoveredDevices.get(i);
    if (di.getName().toLowerCase().trim().equals(
toPairWithEditText.getText().toString().toLowerCase().trim())) {
        deviceFound = true;
        break;
    }
}
if (deviceFound && di != null && di.getBondState() ==
BluetoothDevice.BOND_NONE)
    setEditText(statusEditText, bluetooth.pair(di));
```

Note that the setEditText function is solely for logging activity. The actual pairing is triggered by the bluetooth.pair(di) function.

Step 5. Connect to the paired device. This can be done by selecting the Connect To Paired Device button. This iterates through a list of the currently paired devices and finds the device with a name matching the name entered in the text box next to the Pair With Device Named button. The device does not need to be paired upon opening the application if it was paired previously. Once the desired device is found, connecting to the device is attempted. This is done with the following lines:

```
List<BluetoothDevice> pairedDevices = bluetooth.getPairedDevices();
```

```

BluetoothDevice di = null;
boolean deviceFound = false;
for (int i = 0; i < pairedDevices.size(); i++) {
    di = pairedDevices.get(i);
    if (di.getName().toLowerCase().trim().equals(
toPairWithEditText.getEditableText().toString().toLowerCase().trim())) {
        deviceFound = true;
        break;
    }
}
if (deviceFound && di != null) {
    bluetooth.connectToDevice(di);
    setEditText(statusEditText, di.getName() + " Connecting\nPlease Wait For
Connection Status");
}

```

Communicating

Step 1. Ensure that the device is connected. This can be done by following the “Connecting” tutorial above.

Step 2. Enter text to send to the printer in the text box next to the Send button. Then select the Send button. This will send the entered text to the printer using the following line:

```
bluetooth.send(toSendEditText.getText().toString() + "\r\n");
```

Step 3. If the send is successful and the printer responds successfully, the response will appear in the text box below the Clear Response button. Note that if the response contains or is made up of unreadable characters (such as status bytes), those characters will not appear in the text box. Reading the response is handled by the DeviceCallback object. Any time a byte is received from the printer, it is passed to the onCharReceived function. This is handled in the Bluetooth library by starting a thread to constantly check for new data from the printer. When new data is detected, the dedicated thread will call onCharReceived. The DeviceCallback object looks like the following:

```

bluetooth.setDeviceCallback(new DeviceCallback() {
    @Override
    public void onDeviceConnected(BluetoothDevice device) {
        setEditText(statusEditText, "Bluetooth Device Connected: \n " +
device.getName());
    }
    @Override
    public void onDeviceDisconnected(BluetoothDevice device, String message) {
        setEditText(statusEditText, "Bluetooth Device Disconnected: \n " +
device.getName());
    }
    @Override
    public void onCharReceived(char c) {
        appendStringEditText(responseEditText, String.valueOf(c));
    }
    @Override
    public void onError(String message) {
        setEditText(statusEditText, "Bluetooth DeviceCallback Error: \n " +
message);
    }
    @Override

```

```

        public void onConnectError(BluetoothDevice device, String message) {
            setEditText(statusEditText, "Bluetooth Connect Error: \n " +
device.getName() + "\n " + message);
        }
    });

```

All of the event handler functions within the DeviceCallback object are filled with basic activity logging functionality. The onCharReceived function just appends the received character to the text already in the response text box. The dedicated thread that reads new data looks like the following:

```

try {
    char c;
    while(true) {
        c = (char)input.read();
        if (c == -1)
            break;
        if(deviceCallback != null){
            final char cCopy = c;
            ThreadHelper.run(runOnUiThread, activity, new Runnable() {
                @Override
                public void run() { deviceCallback.onCharReceived(cCopy); }
            });
        }
    }
} catch (final IOException e) {
    connected=false;
    if(deviceCallback != null){
        ThreadHelper.run(runOnUiThread, activity, new Runnable() {
            @Override
            public void run() {
                deviceCallback.onDeviceDisconnected(device, e.getMessage());
            }
        });
    }
}

```

485BluetoothSample

ENABLE BLUETOOTH

START SCANNING

STOP SCANNING

PAIR WITH DEVICE NAMED

CONNECT TO PAIRED DEVICE

SEND

CLEAR RESPONSE

