Smart Band Application:

package com.example.android.bluetoothlegatt.ble\_service;

import android.annotation.TargetApi;

import android.app.Service;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.bluetooth.BluetoothGatt;

import android.bluetooth.BluetoothGattCallback;

import android.bluetooth.BluetoothGattCharacteristic;

import android.bluetooth.BluetoothGattDescriptor;

import android.bluetooth.BluetoothGattService;

import android.bluetooth.BluetoothManager;

import android.bluetooth.BluetoothProfile;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.content.IntentFilter;

import android.os.Binder;

import android.os.Build;

import android.os.IBinder;

import android.support.v4.content.LocalBroadcastManager;

import android.util.Log;

import com.example.android.bluetoothlegatt.constant.Constants;

import com.example.android.bluetoothlegatt.models.BroadcastData;

import com.example.android.bluetoothlegatt.models.DataPacket;

import java.util.ArrayList;

import java.util.List;

import java.util.Timer;

import java.util.UUID;

/\*\*

\* Service for managing connection and data communication with a GATT server hosted on a

\* given Bluetooth LE device.

\*/

@TargetApi(Build.VERSION\_CODES.JELLY\_BEAN\_MR2)

public class BluetoothLeService extends Service {

private final static String TAG = BluetoothLeService.class.getSimpleName();

private BluetoothManager mBluetoothManager;

private BluetoothAdapter mBluetoothAdapter;

private String mBluetoothDeviceAddress;

private BluetoothGatt mBluetoothGatt;

private int mConnectionState = STATE\_DISCONNECTED;

private static final int STATE\_DISCONNECTED = 0;

private static final int STATE\_CONNECTING = 1;

private static final int STATE\_CONNECTED = 2;

public final static String ACTION\_GATT\_CONNECTED =

"com.example.bluetooth.le.ACTION\_GATT\_CONNECTED";

public final static String ACTION\_GATT\_DISCONNECTED =

"com.example.bluetooth.le.ACTION\_GATT\_DISCONNECTED";

public final static String ACTION\_GATT\_SERVICES\_DISCOVERED =

"com.example.bluetooth.le.ACTION\_GATT\_SERVICES\_DISCOVERED";

public final static String ACTION\_DATA\_AVAILABLE =

"com.example.bluetooth.le.ACTION\_DATA\_AVAILABLE";

public final static String EXTRA\_DATA =

"com.example.bluetooth.le.EXTRA\_DATA";

//CLIENT\_CHARACTERISTIC\_CONFIG

public static final UUID CCCD;

public static final UUID RX\_CHAR\_UUID;

public static final UUID RX\_SERVICE\_UUID;

public static final UUID TX\_CHAR\_UUID;

private static final int FREE = 0;

private static final int SEND\_PACKET\_SIZE = 20;

private boolean final\_packet;

private boolean first\_packet;

private int send\_data\_pointer;

private boolean packet\_send;

private byte[] send\_data;

private int packet\_counter;

public ArrayList<byte[]> data\_queue;

static {

CCCD = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");

RX\_SERVICE\_UUID = UUID.fromString("6e400001-b5a3-f393-e0a9-e50e24dcca9e");

RX\_CHAR\_UUID = UUID.fromString("6e400002-b5a3-f393-e0a9-e50e24dcca9e");

TX\_CHAR\_UUID = UUID.fromString("6e400003-b5a3-f393-e0a9-e50e24dcca9e");

}

private int ble\_status;

private boolean sendingStoredData;

private Timer mTimer;

private DataFromActivityReceiver dataFromActivityReceiver;

public BluetoothLeService() {

this.bleDataHandler = new BleDataHandler();

this.send\_data\_pointer = FREE;

this.packet\_counter = FREE;

this.packet\_send = false;

this.ble\_status = FREE;

this.data\_queue = new ArrayList();

}

@Override

public void onCreate() {

super.onCreate();

if (this.dataFromActivityReceiver == null) {

this.dataFromActivityReceiver = new DataFromActivityReceiver();

LocalBroadcastManager.getInstance(this).registerReceiver(this.dataFromActivityReceiver, makeGattUpdateIntentFilter());

}

}

class DataFromActivityReceiver extends BroadcastReceiver {

DataFromActivityReceiver() {

}

public void onReceive(Context context, Intent intent) {

if (intent.getAction().equals(BroadcastCommand.DATA\_RECEIVED\_FROM\_ACTIVITY)) {

BroadcastData bData = (BroadcastData) intent.getSerializableExtra(BroadcastData.keyword);

if (bData.commandID == 0) {

} else if (bData.commandID == 10) {

Log.d(BluetoothLeService.TAG, "BLE\_RECEIVE\_DATA");

BluetoothLeService.this.BLE\_send\_data\_set((byte[]) bData.data, false);

}

}

}

}

private static IntentFilter makeGattUpdateIntentFilter() {

IntentFilter intentFilter = new IntentFilter();

intentFilter.addAction(BroadcastCommand.DATA\_RECEIVED\_FROM\_ACTIVITY);

return intentFilter;

}

// Implements callback methods for GATT events that the app cares about. For example,

// connection change and services discovered.

private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {

@Override

public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {

String intentAction;

if (newState == BluetoothProfile.STATE\_CONNECTED) {

intentAction = ACTION\_GATT\_CONNECTED;

mConnectionState = STATE\_CONNECTED;

broadcastUpdate(intentAction);

Log.i(TAG, "Connected to GATT server.");

// Attempts to discover services after successful connection.

Log.i(TAG, "Attempting to start service discovery:" +

mBluetoothGatt.discoverServices());

} else if (newState == BluetoothProfile.STATE\_DISCONNECTED) {

intentAction = ACTION\_GATT\_DISCONNECTED;

mConnectionState = STATE\_DISCONNECTED;

Log.i(TAG, "Disconnected from GATT server.");

broadcastUpdate(intentAction);

}

}

@Override

public void onServicesDiscovered(BluetoothGatt gatt, int status) {

if (status == BluetoothGatt.GATT\_SUCCESS) {

enableTXNotification();

broadcastUpdate(ACTION\_GATT\_SERVICES\_DISCOVERED);

List<BluetoothGattService> services = gatt.getServices();

Log.i("onServicesDiscovered", services.toString());

} else {

Log.w(TAG, "onServicesDiscovered received: " + status);

}

}

@Override

public void onCharacteristicRead(BluetoothGatt gatt,

BluetoothGattCharacteristic characteristic,

int status) {

if (status == BluetoothGatt.GATT\_SUCCESS) {

byte[] data = characteristic.getValue();

StringBuilder stringBuilder = new StringBuilder(data.length);

int length = data.length;

for (int i = 0; i < length; i++) {

stringBuilder.append(String.format("%02X ", new Object[]{Byte.valueOf(data[i])}));

}

broadcastUpdate(ACTION\_DATA\_AVAILABLE, characteristic);

Log.d("onCharacteristicRead", stringBuilder.toString());

}

}

@Override

public void onCharacteristicChanged(BluetoothGatt gatt,

BluetoothGattCharacteristic characteristic) {

BluetoothLeService.this.broadcastUpdate(ACTION\_DATA\_AVAILABLE, characteristic);

}

@Override

public void onCharacteristicWrite(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic, int status) {

}

@Override

public void onDescriptorWrite(BluetoothGatt gatt, BluetoothGattDescriptor descriptor, int status) {

}

};

private void BLE\_send\_data\_set(byte[] data, boolean retry\_status) {

// if (this.ble\_status == 0 && this.mConnectionState == STATE\_CONNECTED) {

this.ble\_status = STATE\_CONNECTING;

if (this.data\_queue.size() != 0) {

this.send\_data = this.data\_queue.get(FREE);

this.sendingStoredData = false;

} else {

this.send\_data = data;

}

this.packet\_counter = FREE;

this.send\_data\_pointer = FREE;

this.first\_packet = true;

BLE\_data\_send();

if (this.data\_queue.size() != 0) {

this.data\_queue.remove(FREE);

}

if (this.data\_queue.size() == 0 && this.mTimer != null) {

this.mTimer.cancel();

}

// } else if (!this.sendingStoredData) {

// this.data\_queue.add(data);

//// start\_timer();

// Intent intent = new Intent(BroadcastCommand.ACTION\_BLE\_SEND\_REQUEST\_DENIED);

// BroadcastData bData = new BroadcastData();

// bData.data = null;

// intent.putExtra(BroadcastData.keyword, bData);

// LocalBroadcastManager.getInstance(this).sendBroadcast(intent);

// } else if (!retry\_status) {

// this.data\_queue.add(data);

// }

}

public static String byte2HexStr(byte[] b) {

// String stmp = BuildConfig.VERSION\_NAME;

String stmp = "1.0";

StringBuilder sb = new StringBuilder("");

for (int n = FREE; n < b.length; n += STATE\_CONNECTING) {

String str;

// stmp = Integer.toHexString(b[n] & BallSpinFadeLoaderIndicator.ALPHA);

stmp = Integer.toHexString(b[n]);

if (stmp.length() == STATE\_CONNECTING) {

str = Constants.VIA\_RESULT\_SUCCESS + stmp;

} else {

str = stmp;

}

sb.append(str);

// sb.append(CommonConsts.SPACE);

}

return sb.toString().toUpperCase().trim();

}

private void BLE\_data\_send() {

int err\_count = FREE;

while (!this.final\_packet) {

byte[] temp\_buffer;

int send\_data\_pointer\_save = this.send\_data\_pointer;

boolean first\_packet\_save = this.first\_packet;

int i;

if (this.first\_packet) {

if (this.send\_data.length - this.send\_data\_pointer > SEND\_PACKET\_SIZE) {

temp\_buffer = new byte[SEND\_PACKET\_SIZE];

for (i = FREE; i < SEND\_PACKET\_SIZE; i += STATE\_CONNECTING) {

temp\_buffer[i] = this.send\_data[this.send\_data\_pointer];

this.send\_data\_pointer += STATE\_CONNECTING;

}

} else {

temp\_buffer = new byte[(this.send\_data.length - this.send\_data\_pointer)];

for (i = FREE; i < temp\_buffer.length; i += STATE\_CONNECTING) {

temp\_buffer[i] = this.send\_data[this.send\_data\_pointer];

this.send\_data\_pointer += STATE\_CONNECTING;

}

this.final\_packet = true;

}

this.first\_packet = false;

} else {

if (this.send\_data.length - this.send\_data\_pointer >= SEND\_PACKET\_SIZE) {

temp\_buffer = new byte[SEND\_PACKET\_SIZE];

temp\_buffer[FREE] = (byte) this.packet\_counter;

for (i = STATE\_CONNECTING; i < SEND\_PACKET\_SIZE; i += STATE\_CONNECTING) {

temp\_buffer[i] = this.send\_data[this.send\_data\_pointer];

this.send\_data\_pointer += STATE\_CONNECTING;

}

} else {

this.final\_packet = true;

temp\_buffer = new byte[((this.send\_data.length - this.send\_data\_pointer) + STATE\_CONNECTING)];

temp\_buffer[FREE] = (byte) this.packet\_counter;

for (i = STATE\_CONNECTING; i < temp\_buffer.length; i += STATE\_CONNECTING) {

temp\_buffer[i] = this.send\_data[this.send\_data\_pointer];

this.send\_data\_pointer += STATE\_CONNECTING;

}

}

this.packet\_counter += STATE\_CONNECTING;

}

this.packet\_send = false;

boolean status = writeRXCharacteristic(temp\_buffer);

Log.d("lq", "send:" + byte2HexStr(temp\_buffer) + " packet\_counter:" + this.packet\_counter);

if (!status && err\_count < 3) {

err\_count += STATE\_CONNECTING;

try {

Thread.sleep(50);

} catch (InterruptedException e) {

}

Log.e(TAG, "writeRXCharacteristic false");

this.send\_data\_pointer = send\_data\_pointer\_save;

this.first\_packet = first\_packet\_save;

this.packet\_counter--;

}

for (int wait\_counter = FREE; wait\_counter < 5 && !this.packet\_send; wait\_counter += STATE\_CONNECTING) {

try {

Thread.sleep(1);

} catch (InterruptedException e2) {

}

}

}

this.final\_packet = false;

this.ble\_status = FREE;

}

public void enableTXNotification() {

BluetoothGattService RxService = this.mBluetoothGatt.getService(RX\_SERVICE\_UUID);

if (RxService == null) {

// showMessage("Rx service not found!");

// broadcastUpdate(BroadcastCommand.DEVICE\_DOES\_NOT\_SUPPORT\_UART);

return;

}

BluetoothGattCharacteristic TxChar = RxService.getCharacteristic(TX\_CHAR\_UUID);

if (TxChar == null) {

// showMessage("Tx charateristic not found!");

// broadcastUpdate(BroadcastCommand.DEVICE\_DOES\_NOT\_SUPPORT\_UART);

return;

}

Log.d("Moon", "----- enbale tx -----");

this.mBluetoothGatt.setCharacteristicNotification(TxChar, true);

BluetoothGattDescriptor descriptor = TxChar.getDescriptor(CCCD);

descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);

this.mBluetoothGatt.writeDescriptor(descriptor);

}

/\*\*

\* @param action

\* @param dataf

\*/

@Deprecated

public void broadcastUpdate(String action, long dataf) {

Intent intent = new Intent(action);

intent.putExtra(action, dataf);

sendBroadcast(intent);

}

/\*\*

\* @param action

\*/

private void broadcastUpdate(final String action) {

final Intent intent = new Intent(action);

sendBroadcast(intent);

}

/\*\*

\* @param action

\* @param data

\*/

@Deprecated

private void broadcastUpdate(final String action,

final String data) {

final Intent intent = new Intent(action);

if (ACTION\_DATA\_AVAILABLE.equals(action)) {

intent.putExtra(EXTRA\_DATA, data);

} else {

intent.putExtra(action, data);

}

sendBroadcast(intent);

}

private BleDataHandler bleDataHandler;

private void broadcastUpdate(String action, BluetoothGattCharacteristic characteristic) {

Intent intent = new Intent(action);

if (!TX\_CHAR\_UUID.equals(characteristic.getUuid())) {

return;

}

// if (this.ble\_status == 0 || this.ble\_status == STATE\_CONNECTED) {

this.ble\_status = STATE\_CONNECTED;

byte[] received = characteristic.getValue();

Log.d("lq", "received:" + byte2HexStr(received));

Log.d("lq", "received:" + bytesToByteString(received));

if (this.bleDataHandler.add\_data(received)) {

int packet\_status = this.bleDataHandler.check\_packet();

if (packet\_status == 0) {

DataPacket dataPacket = this.bleDataHandler.get\_packet();

if (dataPacket != null) {

Log.d("Data", "Package Data: " + bytesToByteString(dataPacket.data));

BroadcastData bData = new BroadcastData(2);

bData.setReceives(received);

bData.data = dataPacket;

intent.putExtra(BroadcastData.keyword, bData);

sendBroadcast(intent);

}

this.ble\_status = FREE;

return;

} else if (packet\_status != STATE\_CONNECTING && packet\_status == STATE\_CONNECTED) {

this.ble\_status = FREE;

return;

} else {

return;

}

}

// this.bleDataHandler.clear\_packet();

this.ble\_status = FREE;

// } else if (this.ble\_status == STATE\_CONNECTING) {

// if (this.final\_packet) {

// this.final\_packet = false;

// this.ble\_status = FREE;

// }

// if (characteristic.getValue().length == STATE\_CONNECTING) {

// this.ble\_status = FREE;

// } else {

// this.ble\_status = FREE;

// }

// }

}

private String bytesToByteString(byte[] bytes) {

String btyesString = "";

for (int i = 0; i < bytes.length; i++) {

btyesString += " " + bytes[i];

}

return btyesString;

}

private String bytesToByteString(ArrayList<Byte> bytes) {

String btyesString = "";

for (int i = 0; i < bytes.size(); i++) {

btyesString += " " + bytes.get(i);

}

return btyesString;

}

public class LocalBinder extends Binder {

public BluetoothLeService getService() {

return BluetoothLeService.this;

}

}

@Override

public IBinder onBind(Intent intent) {

return mBinder;

}

@Override

public boolean onUnbind(Intent intent) {

// After using a given device, you should make sure that BluetoothGatt.close() is called

// such that resources are cleaned up properly. In this particular example, close() is

// invoked when the UI is disconnected from the Service.

close();

return super.onUnbind(intent);

}

private final IBinder mBinder = new LocalBinder();

/\*\*

\* Initializes a reference to the local Bluetooth adapter.

\*

\* @return Return true if the initialization is successful.

\*/

public boolean initialize() {

// For API level 18 and above, get a reference to BluetoothAdapter through

// BluetoothManager.

if (mBluetoothManager == null) {

mBluetoothManager = (BluetoothManager) getSystemService(Context.BLUETOOTH\_SERVICE);

if (mBluetoothManager == null) {

Log.e(TAG, "Unable to initialize BluetoothManager.");

return false;

}

}

mBluetoothAdapter = mBluetoothManager.getAdapter();

if (mBluetoothAdapter == null) {

Log.e(TAG, "Unable to obtain a BluetoothAdapter.");

return false;

}

return true;

}

/\*\*

\* Connects to the GATT server hosted on the Bluetooth LE device.

\*

\* @param address The device address of the destination device.

\* @return Return true if the connection is initiated successfully. The connection result

\* is reported asynchronously through the

\* {@code BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGatt, int, int)}

\* callback.

\*/

public boolean connect(final String address) {

if (mBluetoothAdapter == null || address == null) {

Log.w(TAG, "BluetoothAdapter not initialized or unspecified address.");

return false;

}

// Previously connected device. Try to reconnect.

if (mBluetoothDeviceAddress != null && address.equals(mBluetoothDeviceAddress)

&& mBluetoothGatt != null) {

Log.d(TAG, "Trying to use an existing mBluetoothGatt for connection.");

if (mBluetoothGatt.connect()) {

mConnectionState = STATE\_CONNECTING;

return true;

} else {

return false;

}

}

final BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);

if (device == null) {

Log.w(TAG, "Device not found. Unable to connect.");

return false;

}

// We want to directly connect to the device, so we are setting the autoConnect

// parameter to false.

mBluetoothGatt = device.connectGatt(this, false, mGattCallback);

// Log.d(TAG, String.valueOf(device.createBond()));

Log.d(TAG, "Trying to create a new connection.");

mBluetoothDeviceAddress = address;

mConnectionState = STATE\_CONNECTING;

return true;

}

/\*\*

\* Disconnects an existing connection or cancel a pending connection. The disconnection result

\* is reported asynchronously through the

\* {@code BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGatt, int, int)}

\* callback.

\*/

public void disconnect() {

if (mBluetoothAdapter == null || mBluetoothGatt == null) {

Log.w(TAG, "BluetoothAdapter not initialized");

return;

}

mBluetoothGatt.disconnect();

}

/\*\*

\* After using a given BLE device, the app must call this method to ensure resources are

\* released properly.

\*/

public void close() {

if (mBluetoothGatt == null) {

return;

}

mBluetoothGatt.close();

mBluetoothGatt = null;

}

/\*\*

\* Request a read on a given {@code BluetoothGattCharacteristic}. The read result is reported

\* asynchronously through the {@code BluetoothGattCallback#onCharacteristicRead(android.bluetooth.BluetoothGatt, android.bluetooth.BluetoothGattCharacteristic, int)}

\* callback.

\*

\* @param characteristic The characteristic to read from.

\*/

public void readCharacteristic(BluetoothGattCharacteristic characteristic) {

if (mBluetoothAdapter == null || mBluetoothGatt == null) {

Log.w(TAG, "BluetoothAdapter not initialized");

return;

}

mBluetoothGatt.readCharacteristic(characteristic);

}

/\*\*

\* Retrieves a list of supported GATT services on the connected device. This should be

\* invoked only after {@code BluetoothGatt#discoverServices()} completes successfully.

\*

\* @return A {@code List} of supported services.

\*/

public List<BluetoothGattService> getSupportedGattServices() {

if (mBluetoothGatt == null) return null;

return mBluetoothGatt.getServices();

}

public BluetoothGatt getBluetoothGatt() {

return mBluetoothGatt;

}

@Deprecated

public void writeCharacteristic(BluetoothGattCharacteristic characteristic) {

if (this.mBluetoothAdapter == null || this.mBluetoothGatt == null) {

} else {

this.mBluetoothGatt.writeCharacteristic(characteristic);

}

}

@Deprecated

public boolean writeRXCharacteristic(String serviceUUID, String charactersticUUID, byte[] value) {

BluetoothGattService RxService = null;

if (this.mBluetoothGatt != null) {

try {

RxService = this.mBluetoothGatt.getService(UUID.fromString(serviceUUID));

} catch (Exception e) {

e.printStackTrace();

}

}

if (RxService == null) {

return false;

}

BluetoothGattCharacteristic RxChar = RxService.getCharacteristic(UUID.fromString(charactersticUUID));

if (RxChar == null) {

return false;

}

RxChar.setValue(value);

return this.mBluetoothGatt.writeCharacteristic(RxChar);

}

@Deprecated

public boolean writeRXCharacteristic(BluetoothGattCharacteristic bluetoothGattCharacteristic, byte[] value) {

BluetoothGattService RxService = null;

if (this.mBluetoothGatt == null) {

return false;

}

bluetoothGattCharacteristic.setValue(value);

return this.mBluetoothGatt.writeCharacteristic(bluetoothGattCharacteristic);

}

@Deprecated

public boolean writeUDCharacteristic(String serviceUUID, String charactersticUUID, byte[] value) {

BluetoothGattService RxService = null;

try {

RxService = this.mBluetoothGatt.getService(UUID.fromString(serviceUUID));

} catch (Exception e) {

e.printStackTrace();

}

if (RxService == null) {

return false;

}

BluetoothGattCharacteristic RxChar = RxService.getCharacteristic(UUID.fromString(charactersticUUID));

if (RxChar == null) {

return false;

}

RxChar.setValue(value);

RxChar.setWriteType(1);

boolean status = this.mBluetoothGatt.writeCharacteristic(RxChar);

return status;

}

public boolean writeRXCharacteristic(byte[] value) {

if (mBluetoothGatt != null) {

BluetoothGattService RxService = this.mBluetoothGatt.getService(RX\_SERVICE\_UUID);

if (RxService == null) {

// showMessage("Rx service not found!");

// broadcastUpdate(BroadcastCommand.DEVICE\_DOES\_NOT\_SUPPORT\_UART);

return false;

}

BluetoothGattCharacteristic RxChar = RxService.getCharacteristic(RX\_CHAR\_UUID);

if (RxChar == null) {

// showMessage("Rx charateristic not found!");

// broadcastUpdate(BroadcastCommand.DEVICE\_DOES\_NOT\_SUPPORT\_UART);

return false;

}

RxChar.setValue(value);

boolean status = this.mBluetoothGatt.writeCharacteristic(RxChar);

mBluetoothGatt.setCharacteristicNotification(RxChar, true);

Log.d("lq", "write TXchar - status=" + status);

return status;

}

return false;

}

public boolean setCharacteristicNotification(String serviceUUID, String characteristicUUID, boolean enabled) {

if (this.mBluetoothAdapter == null || this.mBluetoothGatt == null) {

return false;

}

BluetoothGattService RxService = this.mBluetoothGatt.getService(UUID.fromString(serviceUUID));

if (RxService == null) {

return false;

}

BluetoothGattCharacteristic TxChar = RxService.getCharacteristic(UUID.fromString(characteristicUUID));

if (TxChar == null) {

return false;

}

boolean status = this.mBluetoothGatt.setCharacteristicNotification(TxChar, enabled);

return status;

}

/\*\*

\* Enables or disables notification on a give characteristic.

\*

\* @param characteristic Characteristic to act on.

\* @param enabled If true, enable notification. False otherwise.

\*/

public boolean setCharacteristicNotification(BluetoothGattCharacteristic characteristic,

boolean enabled) {

if (mBluetoothAdapter == null || mBluetoothGatt == null) {

Log.w(TAG, "BluetoothAdapter not initialized");

return false;

}

// This is specific to Heart Rate Measurement.

// if (UUID\_HEART\_RATE\_MEASUREMENT.equals(characteristic.getUuid())) {

// BluetoothGattDescriptor descriptor = characteristic.getDescriptor(

// UUID.fromString(SampleGattAttributes.CLIENT\_CHARACTERISTIC\_CONFIG));

// descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);

// mBluetoothGatt.writeDescriptor(descriptor);

// }

// if (UUID\_CHAR10.equals(characteristic.getUuid())) {

// List<BluetoothGattDescriptor> bluetoothGattDescriptors = characteristic.getDescriptors();

// if (bluetoothGattDescriptors.size() > 0) {

// BluetoothGattDescriptor descriptor = bluetoothGattDescriptors.get(0);

// descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);

// mBluetoothGatt.writeDescriptor(descriptor);

// }

// }

return mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);

}

public BluetoothGattCharacteristic getBluetoothGattCharacteristic(String serviceUUID, String charUUID) {

if (mBluetoothGatt == null) {

return null;

}

return mBluetoothGatt.getService(UUID.fromString(serviceUUID)).getCharacteristic(UUID.fromString(charUUID));

}

}