User's Guide

for

SmartGlove Android Application

Version 1.0

Prepared by Matthew Constant

Wearable Biosensing Lab

November 12, 2017

Contents

1 Introduction

1.1 Purpose

This application is being developed in order to act as a gateway between the SmartGlove e-Textile device and the secure server on which the data collected is to be stored and viewed from. By collecting this data from the SmartGlove device in real-time, the application has the ability to both guide the participant through his/her exercise as well as securely store this data in a server to be accessed at a later time by either the doctor or patient. In addition, this application will allow users to easily visualize the data that is stored on the server in a simple and intuitive way.

1.2 Application Requirements

This application must connect to the SmartGlove device with as little user interaction as possible. This includes filtering devices out when scanning for nearby BLE devices, as well as automatically reconnecting to previously connected devices. This application must also be able to send the collected data to a server via MQTT. Lastly, the application needs to be able to connect to the remote server in order to retreive and visualize a patient's data in the application.

1.3 Implementation

This application has implemented these requirements using the native Android BLE API and the open source MPAndroid Charts library. (The MQTT portion of this application is still under development). Developers can interact with the SmartGlove using the BleConnectionService class. This class allows users the ability to connect to, disconnect from, discover services, read, write, and be notified. The data retrieved is then broadcast using the BleUpdateReceiver. This means that any class listening via a BleUpdateReceiver can collect information from the BleConnectionService. This approach allows for the highest level of abstraction as well as the least impactful on the User's program. All actions supported by the BleConnectionService are made available by static methods in the BleConnectionService. Another layer of abstraction is made by using the BluetoothLeModel class to represent the BLE device to the user. The user will create a BluetoothLeModel and use this to send and receive data from the BleConnectionService.

1.4 Resources

2 Overview

2.1 BluetoothLeModel

Rather than interact directly with the BluetoothGatt class, the user will only need to use the BluetoothLeModel. This is a much more simplified class that still allows the User to take full advantage of the SmartGlove device. The BluetoothLeModel contains the BLE Device address, as well as all of its services and characteristics. For example, in order to connect to a nearby BLE device the user can simpy create a BluetoothLeModel with its Bluetooth Address using the following method: BluetoothLeModel.CREATE(String BT_ADDR). The BluetoothLeModel can allows be created with all of its services and characteristics by first discovering the devices services and adding this list to the create method. For example:
$BleConnectionService.DISCOVER_SERVICES(Context, BluetoothLeModel)$
...
$BluetoothLeModel.CREATE(StringBT_ADDR, List < BluetoothGattServices >)$;

2.2 Sending Commands to BleConnectionService

Users can interact with the SmartGlove by sending ´Actions´ to the BleConnectionService. These actions are made available with static methods. For example, in order to connect to a nearby BLE device, the user can simply call BleConnectionService(Context, BluetoothLeModel). The following commands are currently implemented by the BleConnectionService:
CONNECT
DISCONNECT
DISCOVER_SERVICES
REQUEST_READ

2.3 Receiving Updates and Data from BleConnectionService

In order to retrieve information from the BleConnectionService, the user must implement the BleUpdateReceiver. All data and updates from the BleConnectionService are transmitted through this Receiver. Updates includes when a device is connected to or disconnected from, the services of a device are discovered, and when the device is read from or written to. Each update will include the corresponding BluetoothLeModel,

providing the user with the Device address and the most current values for each characteristic.

2.4 Adding Features to BleConnectionService

Implementing new features to the BleConnectionService is a very straight-forward process. First, the action must be defined in the BleActions enum. Next, a static method must be implemented in the BleConnectionService to provide an interface for users to use this feature (Please follow the pattern of all other features, meaning the static method should simply generate the intent needed and send that intent to the service). Third, add the action into the switch statement in the onStartCommand method in BleConnectionService. This should do some preliminary error checking and then call the corresponding method related to this action. The last step is to implement this method that is related to the action. If this feature must also send data or updates back, it must be sent using the BleUpdate Broadcast Receiver.

2.5 Areas to Improve