Welcome back. I'm Alan Hawse. This is Cypress Academy. Now we're going to dig through the Android app used to control our PSoC BLE LED and CapSense project. I'm using Android Studio 2.1 in these videos but other versions should work just fine.

Project Setup

We'll start by creating a new project called BLE101. I used a Company Domain of "academy.cypress.com". This can be anything but it needs to be unique for every application. It's recommended to use reverse internet notation for your own domain to guarantee uniqueness.

Select "Phone and Tablet" and then a "Minimum SDK" of "API 18: Android 4.3 (Jelly Bean)". This is the first Android release that supports BLE so there is no reason to use a lower API. We will chose an empty activity and keep all of the rest of the settings at default for this activity as well as the layout names.

Once it finishes, Android Studio is nice enough to give us a "Hello World" application that we can modify for our own purposes. In fact, if you have an Android device, you can connect it to the computer in debug mode, program the application to it, and you will see the "Hello World" displayed on the screen.

If your phone does not connect in debug mode, you may have to enable it. Typically you do this by going to "Settings > About Phone". Then you scroll down to the "Build Number" and tap on it seven times – not six, not five, but seven. Once you do this you will see "Developer Options" in the "System" menu. You can open "Developer Options" and then turn on "USB Debugging". This will allow your phone to attach to the debugger inside of the Android Studio.

Introduction

The project will contain 4 main parts that we will discuss briefly now and then in more detail in the rest of this video. Over on the left side if you click on "Projects" it will organize the files so that you can find them easier.

Manifest

The first section is the manifest. The “AndroidManifest.xml” file describes the application in general terms – listing each of the activities and services that the application has as well as some of the overall properties of the application.

Layout

The second section is the user interface or the layout. This describes what the screen of the app will look like. You can find this file under “res > layout > activity_main.xml”. Note that an application can have many different layouts – one for each activity. In this case we will only have one.

Activity

The final section is the MainActivity file which can be found under the “java” path. In Android, an Activity is a part of an application - it’s specifically and technically a java class - that allows users to interact with the device in a specific way. The activity interacts with and controls the layout so that the application can respond to user input and provide information to the user on the screen.

Service

The next section is Service. The new project doesn’t have one yet, but we will eventually create a Service class which will show up in the java folder. This will contain a model of our BLE device, the same one that I discussed in the previous video.

Details

Manifest (AndroidManifext.xml)

Alright, let’s dig into the details starting with the manifest, or AndroidManifest.xml. In our case the manifest file that comes as a default has most of what we need but there’s a few permissions we need so that Bluetooth will work properly. For Android 6 and above – Marshmallow and above - the course location permission is required for BLE scanning to work. These lines are

added before the application description in the manifest file. They can also be found in the provided example project which you can get on Cypress.com.

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<!--  Location permission required for android 6.0 (Marshmallow) -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Layout (activity_main.xml)

The next file activity_main.xml. This is the layout file for the main activity. For this layout, we'll blow away all of the defaults and add our own layout consisting of the buttons, the switches, and the text views as we demonstrated in the previous video. The layout can be created either by using graphical entry (Design tab) or by writing the XML code directly (Text tab).

We will not go through the details of the layout but it can be found in the provided example code.

Note, the names displayed on the buttons, sliders and CapSense Value are not hard-coded in the layout file. Rather, they are referenced from a strings file which can be found in res > values > strings.xml. This is done so the app can easily be localized to support multiple languages. The only thing needed is to create a strings file for each language that you wish to support. The phone's language selection will automatically take the string from the appropriate file. For example, if you wanted to support French, you would create a file called "res > values-fr > strings_fr.xml" with the French equivalent for each string.

Service (PSoCCapSenseLedService.java)

Before covering the activity - which will bring everything together – we'll discuss the service that will model our PSoC device. To create a new service, right click on "java > com.cypress.academy.ble101 > New > Service > Service". We will call the new service "PSoCCapSenseLedService" and turn of the "Exported" since other applications will not need access to it.

Note: When we create the service this way, it's automatically added for us to the manifest file.

Once the service is created, we can open the PSoCCapSenseLedService.java file and start adding the required BLE functionality. Again, we won't go through all of the details here but the code is provided for you to examine. The basic contents are:

1. We first create variables for BLE objects such as the BLE manager, the adapter, the scanner, etc.
2. We then create variables for the UUID for our services and characteristics.
3. Then we set up a binder that will allow the main activity to bind to the service.
4. Then we set up helper methods that will allow the main activity to interact with the various functions of the BLE model such as initialize, scan, connect, disconnect, discover services, disconnect connection, and read/write data.
5. Then we set up callbacks for each of the BLE actions. These callbacks can broadcast updates to the main activity when necessary. The broadcastUpdate method in turn calls "sendBroadcast" which is picked up by the broadcast receiver in the main activity and I'll talk more about that a little bit later in this video.

Note: In order to support multiple Android versions seamlessly, Google has given us @TargetAPI that will be used before the service class definitions. This is so that the APIs introduced in LOLLIPOP and later are allowed in the code. It's the writer's responsibility to ensure that any APIs not available prior to that API are not called on devices that don't support them. This technique is used so that different BLE scan methods can be used on phones running different versions of the operating systems.

Activity (MainActivity.java)

The next major area is the MainActivity.java. This is the main Java class. It controls the main activity behavior of the application. Specifically, it controls the layout as well as interacting with the BLE model. That is, each time the user presses a button or a switch on the screen, the activity senses the action and calls the appropriate method. Likewise, whenever the BLE activity such as a notification

occurs, the activity is notified using the broadcast receiver and it then takes the appropriate action.

The organization of this file is as follows:

1. Variables are set up for the objects within the layout such as the buttons, the switches, the text views, etc.
2. Variables are then set up for the BLE service and a new serviceConnection object is created.
3. When the activity starts, the onCreate method is called. This method draws the layout, and registers the methods that will be called when the switches are toggled, such as the LED switch and the CapSense notification switch.
4. The onResume method is the called. It registers the broadcast receiver for the BLE events that we are interested in.
5. Other methods such as onPause, onDestroy which are used to clean up resources once they're no longer needed, like when the application quits.
6. A set of helper methods are created to handle each of the button presses. For example, "startBluetooth" is called when the "Start Bluetooth" button is pressed. This method gets the BLE adapter object, and binds it to the service connection object. It then disables the "Start Bluetooth" button and enables the "Search for Device" Button.
7. Once the service has been connected, the "onServiceConnected" method is called - this is part of the serviceConnection object - and that provides us with the service object pointer that we will use for all of the BLE communications.
8. There are four similar methods for each of the buttons such as "searchBluetooth", "connectBluetooth", "discoverServices", and "Disconnect". Each of these call the appropriate methods from the service that will be established and controlled in the layout. In addition, the methods from the service broadcast messages that are picked up by the broadcast receiver in the main activity. This allows the main activity to know when certain BLE events have completed. For example, each of the buttons is enabled in turn once the function before it has completed. The

main activity knows that the prior function has completed from the broadcast messages it receives from the service.

9. Once the connection is established and we've discovered the services and characteristics from the PSoC 4 BLE, then the LED and CapSense functionality can be exercised. This is done once again using messages from the broadcast receiver. It notifies the main activity whenever the user toggles one of the switches or when a CapSense notification value is sent from the device. The BroadcastReceiver in the main activity takes the appropriate actions based on the message it receives from the service.

That's it. With this Android application, you can now control the LED and read the CapSense value from your own phone. Remember to visit www.cypress.com/BLEAndroidApp to download the completed copies of both the PSoC Creator as well as the Android Studio projects.

In the next videos I'm going to take you through the firmware that runs the BLE robot.