Hello, I'm Alan Hawse. Welcome to the Cypress Academy. In this lesson I'm going to take you through the BLE robot car's Android app. This app is similar to the previous one we created with a few big differences actually:

1. Instead of stopping once we find the first device with the BLE scan, we will continue to search and will create a scrollable list of all of the devices that are advertising that they have the robot control service. This will require a new activity. The new activity will be the first one to start once the app launches so we want to create a project with an activity called "ScanActivity" instead of "MainActivity". We will setup this activity so that a swipe down on the list will cause a refresh of the BLE devices.
2. Once the user selects the device that they want from the list, the app will start a new activity to control the robot, called "ControlActivity". This class will extend the "AppCompatActivity" class just like the ScanActivity class does.
3. We will create a service just like we did in the last app but this time the service will be customized to control the robot rather than the LED/CapSense project.

The steps to create the app are exactly the same as the BLE101 app except that we'll call it "BLE101_robot", I'll call the main empty activity "ScanActivity", and the layout will automatically change its name to "activity_scan" which will be fine. That makes good sense.

Alright, let's dig into the details.

Manifest

First the manifest. The manifest will be created automatically along with the project – Android Studio does that for you. For now we just need to add the same permissions that we did for the previous project in order for the BLE to work in all of the different versions of Android.

Layouts

The next thing is the Layouts. We will ultimately need three layout views for this application – two used by the scan activity, and one used by the control activity.

Scan Layouts

Alright, let's dig into the scan layouts. One of them - activity_scan.xml - was created automatically at the same time the project was but we'll need to modify it to contain a SwipeRefreshLayout which in turn will contain a ListView.

The second one used by the scan activity is called "ble_device_list.xml". This is a layout containing a text view. The text view will hold the BLE device name for any devices that are found during scanning. These text views will be created and added to the ListView as they are found. This new layout can be created by right clicking on "layout > New > Layout Resources File" and providing the name "ble_device_list.xml". The root element of this will be "TextView".

Control Layout

The next layout is the control layout. For controlling the robot, we will need another layout view. Since it will ultimately be associated with another activity, the simplest way to do this is to create a new empty Activity. You can do this by right clicking on java > com.cypress.academy.ble101_robot > New > Activity > Empty Activity. Set the name to "ControlActivity" and leave the rest of the values as their defaults. This class will also extend the AppCompatActivity class.

Once the new activity is created, we will have also have a layout called "activity_control.xml". This is the layout that has the actual robot controls in it. The left and right sliders, and the left and right switches.

Service

Add a new service to the project and call it "PSoCBleRobotService".

This Service is very similar to the one that we created for the LED/CapSense project. It has a few extra functions and I'll be going through them step by step. For example, once the user selects the device that they are interested in – the BLE device - the activity binds the service, initializes, connects, and the does service/characteristic discovery instead of doing them as individual user selections.

In the last project we triggered the operations one by one – connect, scan, etc. In this case, we're going to group all of the operations together so that they're not connected to a user button press, but happen one right after the other automatically.

Scan Activity

To start with, the Scan activity will search for BLE devices and it'll list them on the screen. It will filter the list to only devices that are advertising the correct service UUID. The list will be able to be refreshed by using a down swipe from the top of the screen. As described previously, the activity_scan.xml and ble_device_list.xml files – they are used to display the values to the screen.

The details of the ScanActivity are not shown here but you can read through the provided example project.

Control Activity

The control activity here is similar to the main activity from the previous example. That is, the BLE/CapSense example. It controls the layout and interacts with our BLE model. That is, each time the user interacts with the screen - the activity screen - the activity senses that action and calls the appropriate method in our model. Likewise, whenever the BLE activity such as a notification occurs, the activity is notified using the broadcast receiver scheme provided by Android. It then will take the appropriate action.

The organization of the activity is as follows:

1. The variables are set up for the objects within the layout such as the buttons, and the switches, and the text views. This is just a convenient way to simplify your programming by having the connections to the correct layout devices at your fingertips already initialized in a variable.
2. The next thing you do is you set up the variables for the BLE service and the new serviceConnection.
3. When the activity starts, the onCreate method is called. This method initializes the variables, it draws the layout, and then it binds to the service. It also registers the methods that will be called when the switches are

toggled to turn on the motors or when the motor speed sliders are moved. At this point, you cannot forget to put in the special lines that enable Marshmallow to work correctly. If you forget, the system will work but you won't actually find anything and it'll be really annoying and hard to debug.

4. When the onResume method is the called, it registers the broadcast receiver for the BLE events that we are interested in. Once that is done, we tell the service to connect to the BLE device that was chosen on the list. So, when they press the button on the scrolled list, we tell the model to create a BLE connection.
5. The other methods such as onPause and onDestroy are also created to clean up the resources as they are no longer needed when the application exits or the task is switched out.
6. A couple of other helper methods are created to handle some of the tasks that were needed to control the robot. For example, "scaleSpeed" is used to convert the value from the slider, which ranges from 0 to 20, to the values expected by the robot, from -100 to 100. Likewise, the "enableMotorSwitch" is provided to allow either of the motors to be enabled or disabled depending on its arguments. This simplifies our code some.
7. As with the prior example, communication to and from the service is done using the BroadcastReceiver object which is created in the control activity. It looks for various messages such as connected, or disconnected, or data available. One example would be the tachometer values that are coming back from the PSoC that need to be displayed on the screen.

OK, now we'll do a little demo. The first thing I'll do is I'll start up the app. Away we go. See there's nothing on the screen. That's because the robot is off. I'll turn on the robot. The red light will start blinking. As soon as that does, it shows up on the list. I'll press the button to connect. The light stops blinking on the robot. Now I've got control of both sides of the robot. I'll turn on one of the sides of the robot and give it some power, and look – it starts turning. Cool. So, I'll stop that side, I'll start the other side, and I can control the other motor – it turns the other way. How sweet is that? See its turning 200 RPMs. Then I'll press stop. We are all good.

Back up. Now I can see the robot is there, it should start advertising again – red light – that's good.

OK, that's it. I've made 10 videos about how to build a BLE robot. I started by teaching you how to use CapSense, how to create BLE projects, how to program them into your board, how to write an app that connected to them directly, then I built up a robot, I showed you how to make the firmware that ran in the BLE, then I took you through the Android app development for the robot. So at this point there's no reason why you shouldn't be able to go build your own end to end system.

We have all of the bill of materials for this robot is available on our website. All of the projects that we've talked about are also available on the website. If you've got any questions you're always welcome to email me at alan_hawse@cypress.com or you can talk to our technical support and they're available 24x7 and we can help you out. Good luck with your projects, and good luck with PSoC. Thank you.