PSoC Academy: How to Create a PSoC BLE Android App
Lesson 6: Create the App

Hello. My name is Alan Hawse. Welcome back to Cypress Academy. In this lesson, I'm going to walk you through the creation of an Android app to communicate with your PSoC 4 BLE CapSense LED Board.

In this lesson, I'm not going to try to teach you Android Studio, Java, or XML. If you need to learn these, please go watch the excellent "Android Development for Beginners" videos on Udacity.com or many other videos which are available on those of those topics around on the internet.

First a little background. Android provides a framework for Bluetooth Low Energy (BLE) which was introduced in Android 4.3 (API Level 18). There are seven classes that we will use from that framework: BluetoohManager, BluetoothAdapter, BluetoothLeScanner, BluetoothGatt, BluetoothGattCallback, BluetoothGattCharacteristic, and finally BluetoothGattDescriptor.

1. The BluetoothManager is used to find the Bluetooth System Service for the phone that your application is running on.
2. The BluetoothAdapter represents the phone's Bluetooth adapter. This object is used to interact with the radio. It's found using the getAdapter method from the BluetoothManager object.
3. BluetoothLeScanner is used to find the BLE devices that the phone can interact with. It listens to BLE advertising packets and then notifies you

of their existence. It has a callback function that is invoked each time a BLE device is found.

4. BluetoothGatt is used to enable communication with a BLE device. This object is created when we connect to a BLE device or a BLE peripheral.
5. BluetoothGattCallback is used to register events from the BLE device such as connection state changes, and new data available.
6. BluetoothGattCharacteristic is used for each of the characteristics contained in the BLE device's profile.
7. BluetoothGattDescriptor is used for the additional characteristic attributes. In our case it will be used to contain the notification descriptor for the CapSense characteristic.

Now, let's talk about the way the application moves through each of the states that we use.

One thing I didn't mention up until now is permissions. In Android, Bluetooth requires two specific permissions and for Android 6.0 and later a course location permission. Each of these are included in the manifest file along with another line indicating that the phone has to support BLE for this app. In addition, we check whether or not these permissions have been granted within the app itself and request permission if they have not.

The other thing we do during initialization is set up a callback for BLE events. In Android, this callback is a class we call “BroadcastReceiver”. We register the receiver for specific events like the BLE scan callback, the GATT connection and disconnection, and new data available.

Once the permissions and the broadcast receiver are out of the way, we wait for button clicks. At the beginning, only the Start button is enabled. When the user clicks that button, we create the BluetoothManager and the BluetoothAdapter objects. We then start a “Service” which will represent the Bluetooth device that we will eventually connect to. In Android, a Service is an application component that performs the operations in the background without its own user interface. This is perfect for the Bluetooth device – the main application will interact with it and the user interface. That is, a service will be the “Model” of the BLE device that we connect to. In fact, from this point forward, I’ll refer to the Android Service as the “Model”.

Once these steps are done, we enable the next button which is “Search for Device”. When the user clicks that button, we call a method in the Model to start scanning for BLE devices. In this case, we tell the scan method to look only for devices that contain the BLE service that we are looking for by specifying the service’s UUID. Don’t confuse the service in the BLE device with the Android Service object that we created earlier. The Android Service object is

a background task manager that models the BLE device interface, while the BLE device service describes how the BLE device interacts with the phone. In this case, the BLE service is the custom CapSenseLED service that we created in the earlier lessons.

The scan method has a callback function - whenever that callback is invoked, we stop the BLE scanning and then we broadcast an update that will be received by the broadcast receiver in the main application. The broadcast receiver then enables the next button, “Connect to Device”.

When that button is clicked, a method is called in the Model to connect to the device and to create a BluetoothGATT object. Notice that the blue light stops blinking – that’s because the PSoC firmware we wrote stops blinking the LED when a connection is established.

Once that is done, the BluetoothGattCallback object broadcasts an update to our main application. The broadcast receiver in the main application then enables the next button, “Discover Services and Characteristics”.

That button calls a method in the Model which queries the GATT database for the services and characteristics that it contains. When that completes, we get a GATT callback which creates the BluetoothGattCharacteristic object for the

LED characteristic and the CapSense characteristic in our device and a BluetoothGattDescriptor for the CapSense notification.

Once this is done, we broadcast an update that is once again received by the main application. The main application then enables the LED switch, the CapSense notification switch, as well as the “Disconnect” button.

At this point, the application monitors the two switches. If the LED switch is toggled, the application calls a method in the Model to write the LED characteristic. See when I press the red LED the red LED goes on and off. Likewise, if the CapSense Notify switch is toggled, the application calls a method in the Model to write the descriptor - the CCCD – and it changes it from Notify On to Notify Off. See when I switch it, it says No Touch and when I reach down here on the board I can see that the numbers go up and down, and when I flip the switch back it goes back to Notify Off.

If CapSense notifications are enabled, then we get a GATT callback whenever there’s new data available. Basically, when the CapSense is done it will broadcast the fact that there’s been a change. The GATT callback broadcasts an update once again to the main application receiver which is then able to update the CapSense value on the screen.

Finally, I can press the Disconnect button, which will disconnect the BLE device. Remember that our PSoC firmware goes back to advertising when it's disconnected and we see the blue LED start to blink again. I can then exit from the application.

In the next video, I'll walk you through the app itself and show you all of the parts for creating your own Android app.