**PSoC Academy: How to Create a PSoC BLE iOS App**
**Lesson 10: BLE Robot App**



00:00:08 Hello, I'm Alan Hawse. Welcome to Cypress Academy. In this lesson I'm going to take you through the remote control car's iOS app. This app is a good bit more complicated than the last one I showed you. It has two view controllers. The first view controller is a table view controller that displays all of the devices that the BLE has heard. That class is MainTableViewController.swift. When that class starts it starts the bleLand, which is what I call the BlueToothNeighborhood.

00:00:41 It's all the stuff that's related to the Central Manager. The entire view controller is a table view data source. And all it does is it keeps track of all of the BLE devices that match the services that it has heard. You end up with a table view full of the devices that it has heard. In this case there is only one remote control car so there will only be one thing in the table but I wanted to give you an example of how you could have more than one peripheral connected to the Central at a time.

00:01:12 The second view controller is actually the remote control itself. At the top there are two labels; one for the left tachometer, one for the right tachometer. There is two switches; the left one does an all-stop on the left motor, the right one does an all-stop on the right motor. And then there is a picker view which goes from minus 100 to zero to plus 100. And the right picker view does the same thing.

00:01:46 The other thing that I've done in this example is that I've split the model into three classes. The first class is the RcCar.swift. It's a simple class. It keeps track of what its Bluetooth connection is, and it keeps track of the left and right speed, and the left and right tachometer. The next class is called BlueToothNeighborhood. This class only handles the CBCentral functions.

00:02:16 It doesn't handle the actual connection to the car. Other than that it works exactly the same as the previous class that I showed you. The last class is the BleConnection. This class handles the actual connection to each individual car. So the RC car object will have a Bluetooth Neighborhood that it's associated with and it will have a Bluetooth connection that it's associated with.

00:02:43 In the RCViewController when the view loads up it tells both of the picker views that it's in charge of the data for the two picker views. Then it registers the fact that it's interested in changes for when there is a complete Bluetooth connection, just like we did in the past. When there is a connection that occurs it turns on the left and right picker so that you can change the speed of the car.

00:03:11 It also sets the picker to the middle one which is 10, halfway between minus 100 and plus 100. It's the tenth one, which is zero. It also adds a listener for when there is a disconnect event. So if somehow or the other the device is turned off, the peripheral is turned off, or for whatever reason you go out of range then the

picker views go back to zero and the thing turns off.

00:03:39 The last thing that happens is if the tachometer is updated because of a BLE notification sending out an NS notification then it updates the left and right text values on the screen. The last thing is before the view disappears you need to disable the notifications and you need to disconnect the device.

00:04:05 So if you're connected to a device and you're running it backwards and forwards and stops and starts and all of that jazz and you want to go back to the list of other devices and you press the back button at the top of the navigation controller you need to disconnect the device and stop listening for notifications. The rest of this file just handles the pickers. When the picker values change from minus 100 to plus 100 you need to notify the car's model that those values have changed.

00:04:35 It does that by writing into the public API of the car with the new value of the picker. That's all there is to the RC view controller. Then earlier we went through the Bluetooth Neighborhood. The Bluetooth Neighborhood in this app works almost exactly the same as the previous one. This handles the CB Central Manager and making a connection to the car. The functions are almost exactly the same as we've seen in the past. This time instead of just keeping track of one peripheral at a time I have an array.

00:05:07 And that array is called cars and it just contains one entry per

peripheral that matches the services. I have the same startUpCentralManager function that we had before. I have the same discoverDevices. I have the same connectToDevice. I have the same disconnectDevice. When a peripheral is discovered it searches through the list that we already know about, and if it already has heard of it before, it doesn't do anything.

00:05:36 If it's never heard of it before then it adds to the list and sends out a notification that it's found a new car. This would allow the table view controller to reload its data to add more entries into the table. I have the same disconnect device which allows you to disconnect a BLE connection to a peripheral. I have the same when a connection is done send out a notification that the connection is done. In this case I also start the service discovery immediately.

00:06:06 So rather than having to have the user press connect to device, then start the service discovery, then start the characteristic discovery I do that automatically for them. I have a helper function which will search through the list of devices that we know about and give you a reference to the one that it finds. And then I have the same centralManagerDidUpdateState function which sends out the notification that you're connected.

00:06:32 The last object I have is the BleConnection object. Rather than have the BLE central and the BLE peripheral together in the same object I split the object apart. This object works just like the

peripheral stuff that I had in the previous example. It keeps track of the left and right characteristic, the left and right tachometer. It allows you to discover the services in the peripheral. It allows you to discover the characteristics in the peripheral.

00:07:07 And it allows you to give updates for after you've asked for a notification on the tachometer. It allows you to write the left and right speed. And I've given you a little helper function which allows the left speed write and the right speed write to call one function to actually do that write. The other thing that I did in this example is I put all of the global structure in a globals.swift so you can see the data together.

00:07:36 That file just contains the notifications and the BLE parameters; the UUIDs of the services. That's it. I would encourage you to try this for yourself. You can download the Xcode Project as well as the BLE firmware. And you're welcome to send me emails at alan_hawse@cypress.com and I'll be happy to help you out. Now I'm going to scoot this stuff out of the way and we'll run the car a little bit.

00:08:02 Okay, now I'm going to turn it on, start it up, there it is. It recognizes it so I'm going to start it turning. Now I can stop that motor. Then I can run the other motor the other way. So now if I slow it down then it can start going the other direction. Then I can do it majorly fast. That's it.