PSoC Academy: How to Create a PSoC BLE Android App
Lesson 3: Configure the BLE Component

Welcome back. At this point we've completely configured our schematic. The schematic will define all of the firmware that will run inside of our project. Remember there's two things going on. There will a CapSense slider, and there will be a red LED. The CapSense slider will be able to talk to the Android app. The red LED, you'll be able to turn on and off with the switch on the screen.

So now we need to configure the BLE component, so that it's got those two characteristics in it. We'll start by double clicking the BLE. Now, that we've got the BLE customizer open I need to configure the profile. This is going to be totally a custom, so I'll select custom. The device will act as a GATT server, and the GAP roll will be peripheral.

GATT and GAP are the two the most important concepts in understanding this whole system. You can think of GAP as the way the two devices get connected together. The Android will serve as the GAP central. And the BLE device will serve as the GAP peripheral. The GAP central will talk to the GAP peripheral. The GAP peripheral will advertise that it's around. And the GAP central will hear that it's around and make the connection. I'll show you how to do that in a few video.

As I said, GAP is how devices connect and GATT is how devices trade information. This board, which you're going to configure, will run a GATT

server. Remember, a GATT server is just a database that runs inside of the firmware on the PSoC 4 and it remembers information that the GATT client, which is running on the Android phone will be able to grab information out of. It can read and write from the GATT database that's inside of your GATT server.

At this point we have the generic configuration done for the BLE component. The next thing that we need to do is customize the profiles. This is essentially setting up the GATT database so that the Android phone will be able to read data out of it. PSoC Creator gives you a custom service to start from, though I'm going to start by deleting this service, and I'm going to create my own service. So I'll go to the server, I will need to add a custom service.

I'm going to create a service that has to two characteristics in it. One characteristic will be for the LED, and the other characteristic will be for CapSense. Those two characteristics are going to be grouped together into a custom service. That service is going to be called the ledcapsense service, and it will have its own unique identifier. Those unique identifiers are called universally unique identifiers. The UUID we're going to tell the Android phone app so that it will be able to find devices that have that service. First, I'll change the name of the service. I'm going to called it ledcapsense. This is going to give the ability inside of my firmware to talk to that specific service.

Then I'm going to take the first characteristic, which Creator calls Custom Characteristic, and I'm going to change the name. And the name of this characteristic as I discussed earlier is going to be led. That characteristic is going to represent the state of the LED. This characteristic needs to be able to be read and written remotely, so I'll enable the Write flag and I'll enable the Read flag. I'm not going to use the Custom Descriptor, so I'll just delete it.

The next thing I need is the CapSense characteristic, so I'll add another custom characteristic. Just like before I'll rename it, so that I have sensible names in my APIs. This characteristic for some reason – oh, yeah, because I call it CapSense - I'm going to call it capsense. This characteristic is not going to be writable remotely. That doesn't make any sense. You'll only be able to read it. So I'm going to click the Read button.

The other thing that's interesting and that you'd like to be able to do on the other side would be to be notified when things change. In order to be notified you have to click the Notify flag and that will create another special characteristic called the Client Characteristic Configuration Descriptor – and yes, that's a mouthful – it took me 5 takes on this video to say that correctly - so I like to rename that. I'm going to rename that crazy thing to capsenseccccd.

Now I'm not, once again, going to use the custom descriptor, so I'll just delete it and get it out of the way.

When you're looking at this thing remotely from a GATT browser, like the one built into CySmart, once of the nice things to be able to do is to find a human readable form – a name – for the characteristic. In order to do that you add a special descriptor called the Characteristic User Description. I'm going to add that to the LED, and I'm going to type in something sensible. In this case. I'm typing led uint8. So on the remote side you'll be able to see the name of this characteristic and you'll have a human readable information which should make sense to you.

Now I'll go ahead and do the same thing for CapSense by adding a Characteristic User Description for it as well. I'll call this one capsense uint16. On the remote side I'll know that it gives me an unsigned int16 in human readable form.

The next thing that needs to be configured - and I should have configured it originally - I should have set the type of the capsense service. It needs to be a uint16. The default is uint8. Alright, now I've got that fixed.

So we've got the LED setup - it's a uint8. It has a Characteristic User

Description, which is human readable, OK that's good. We've got the capsense characteristic, which is uint16, that's two bites in length. OK, that's good. It has a CCCD – Client Characteristic Configuration Descriptor. And the last thing is to have a Characteristic User Description called capsense uint16.

Now, I have one more thing to do. Each of these things needs to have a specific 128 bit UUID. This is so the other side – the Android Phone side - will be able to identity each of the characteristics in the service. Creator gives it a default and just for grins I'm going to make it say same thing: blah, blah, blah, blah zero. The LED characteristic is going to be blah, blah, blah, blah one. And the CapSense will be blah, blah, blah, blah two.

At this point we've set up the general information for the component. It's a custom profile. I made its custom profile. It's a GATT server. In other words it's running the database. And it's a GAP peripheral, which means that the central can attach to it

I've set up the profile. I've created the custom service. It's got two custom characteristics. That's all configured. The last thing I need to do is I need to set the GAP settings. Inside of the GAP settings tab there's a couple of things I want to do. First of all, I want to give the device a name. So I'm going to call it capled. Then I need to set up the advertising settings. For this setup I'm just

going to advertise all the time, so I'll turn off the advertising time out switch.

This device, you want to be able to connect to it. I'll advertise all the time when it's not connected. The last thing I'm going to do is inside of the advertising packet that's being sent out every 20 to 30 milliseconds – I want to put some information into it so the other side can easily identify us. The first thing I'm going to put in is the name that I just gave it. That will be accessible inside all of the advertising packets that are coming out every 20 to 30 milliseconds. It will show up on the remote side as something identifiable. I'll be able to read its name easily.

Then the last thing I want to do is I wanted to advertise the service that it has available. On the Android phone side, I'm only going to look for devices that match this specific service. When I get the advertising packet on the Android side I'm only going to match packets that have the right service.

At this point the schematic is completely created. We've got the BLE set up. We've got the CapSense set up. We've got the blinking LED, which will indicate the state. We've got the red LED set up. Now, you need to configure your pins. So go to the design wide resources tab - DWR. The next thing you need to do is you need to connect the CapSense slider to the appropriate pins on the board. In this module that's P2[1] through 2[5]. So: P2[1], P2[2], P2[3], P2[4],

P2[5]. That's cool. All of those things are easy to see on the silkscreen on the board. The next thing I need to do is to connect the blue LED to P3[7], and I need to connect the red LED to P2[6].

All right. At this point the pins are set correctly, the schematic is set up, so to help myself here in a minute with the firmware, I'm going go ahead and do a Generate Application. This will create all of the APIs so that they will be known by Creator which ones you're going to use and that will simplify the firmware development. Now that the application is generated go ahead and go to the next lesson. In the next lesson I'm going to take you through creating the firmware.