PSoC BLE 101: 3. Add a Battery Service and Test with CySmart

My name is Alan Hawse. I'm Vice President of Technical Staff for Solutions and Software at Cypress Semiconductor. I hope you're learning a lot from this video series on PSoC BLE.
In this lesson we will take the Find Me profile you built in the first two lessons and add a Battery Level service. This is a very common thing to do because most Bluetooth Low Energy devices are battery-powered, often with just a CR2032 coin cell.
Starting with our Find Me design, I need to open the BLE component customizer and edit the profile to include the new battery service. In the Profile tab select the Find Me Target and press the Add Service button. Pick out "Battery" from the list and it gets added automatically to the profile. The battery service is one of the many services defined by the BLE SIG. It is also supported for you by PSoC Creator with firmware built into the component.

Now let's initialize the reported level of the battery level characteristic. Under the Battery service is a characteristic called Battery Level. Clicking on it reveals the details of the characteristic and, as the level is a percentage, it makes sense that we should start with a value of 100 to signify a full battery.
That's all we need to do. Just build and program the target.
When I connect to the device from CySmart on my phone I can see the new service with the battery level at 100%. Hey, that's pretty cool.

This demonstrates how easy it is to add the service Ð we did not even write a single line of code because the stack handled everything for us. Though, I have to admit, it's kind of boring because it always reports 100%. So let's change that.

Normally you would use a resistor divider network to create a sensor node to measure the battery with the Analog to Digital Converter. But as that node doesn't exist on the pioneer board I am going to create a synthetic measurement and show you how to update the GATT database with new data. Basically I will use a timer to reduce the battery level by 1% every second.

First of all, let's add a Timer that will generate an interrupt once every second.

I am using the TCPWM-based Timer that I've shown you before. Connect a clock component to the input and change its frequency to the kilohertz range with a value of 12.
In the Timer choose a period of 12000 and make sure you check the TC interrupt. This will cause the Timer to generate an interrupt when it reaches zero - once per second.
Finally, attach an ISR component and give it a friendly name, in this case Timer ISR.
In the main.c, I have to start the Timer and install an interrupt handler.

Moving back up the file I'll write the ISR by using the CY_ISR macro. I'll clear the interrupt and then update the battery level. The level is stored inside the GATT database, which we extended to include the Characteristic of the Battery Level Service. I am going to read the characteristic, modify the level, and write it back into the database.

Remember the GATT Server in your implementation will automatically provide the information to the GATT Client on the other end of the connection.
First, I create two local variables, one for the level and one with the appropriate GATT data structure. In this case it's ÒCY BLE GATT HANDLE VALUE PAIR underscore TÓ Đ and I initialize it.
Remember all of these data structures can be found in the online API Browser, which we talked about in the earlier lesson.

Then I call the "CY BLE GATT S Read Attribute Value API" to extract the information. Yes, I know that is a handful; I just had to say it. But it is important to understand the GATT S part of the API tells you it is part of the GATT server.

Next I modify the battery level by decrementing by 1%, then when it reaches 0% I put it back to 100%. This will give us a characteristic that will change often enough to be interesting. In the real world, you would replace this synthetic measurement with a real battery measurement.
Finally I just have to update the database, so I use the complement to the "Read Attribute" API. To do that; you use "CY BLE GATT S WRITE" instead of READ.

Then I'll build and program the kit just as I have been doing in all the other examples.
As usual I will test the new updates using CySmart. You can see that every time I read the battery level, it has changed by 1%. I've been using CySmart on the phone so far but we also have a PC-based version. It gives you a lot more detail than the phone-based app. With this tool you can browse into the GATT database.
To do this, first start CySmart.
Then, you select the Cypress BLE Dongle and press "connect".

Then you press, "start scan" to see what devices are out there in advertising.
When you see a device you are interested in, select it and then press connect.

Then you press, "Discover all attributes". CySmart will then interrogate the GATT server that you just connected to. It will look through the database and it will create a copy inside of CySmart of exactly what you configured in your peripheral.

Now you click on the Alert Level Characteristic. Then you can type in a new value, either 0, or a 1, or a 2 representing the three alert levels. Once you've selected the number you want, click the write with no response. Notice that if you write a 1 the LED will start blinking.
Each of the Profiles, Services and Characteristics that you defined in the component customizer are available in CySmart.
If you have configured them to be writable then you can write them. If you configured them to be readable then you can read them.
Now we have extended our application by including a Battery Level service. We have now seen applications where the phone sends data to the peripheral with "Write" and where it reads data from the peripheral with "Read".

See if you can replicate this in your own design.
As always you are welcome to email me at alan_hawse@cypress.com with your comments, suggestions, criticisms and questions.
Watch all PSoC Creator 101 lessons at www.cypress.com/creator101