ECEN 5823 Spring 2025
Assignment 5 - BLE Health Thermometer Profile (HTP)

Objective: To take the temperature measured by the Si7021 on the base PCB and communicate it via BLE to the Silicon Labs' **EFR Connect** iPhone or Android phone app.

Note: This assignment will begin with the completed Assignment #4 - I2C temp sensor load power management part 2

References required:

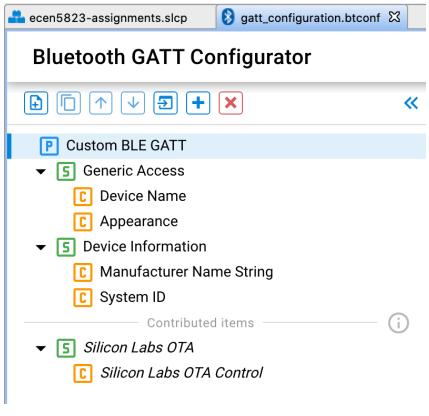
Silicon Labs Bluetooth Software API Reference Manual: see https://docs.silabs.com/bluetooth/latest/

Instructions:

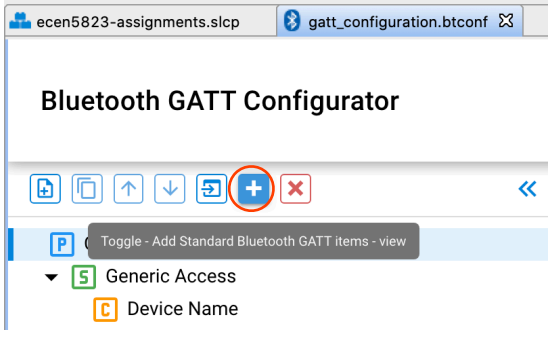
1. Start by creating your assignment repository using the assignment link at https://classroom.github.com/a/DeXk-4v5 . You will not need to clone this from the GitHub repo you just created into a new local directory since you will be starting with your code from the previous assignment. Instead, follow these instructions and run these commands with git bash within your assignment directory. Ensure you are setup to use SSH keys before proceeding, see the instructions from github at https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/ and be sure to use Git Bash For Windows or Cygwin if using a Windows environment. (Cygwin .ssh directory will be different than Git Bash.)

    a. On your local PC, duplicate the folder created in Assignment #4 to a new folder, name it something like ecen5823-assignment5-<username> where <username> is your GitHub username.
    b. Change your current directory to this new folder ecen5823-assignment5-<username> and execute the following git commands.
    c. git remote remove origin
    d. git remote add origin <url>
        i. Where <url> is the URL for the repository created with the link above
        ii. This adds the new submission repository created in the link above as your new origin.
    e. git push origin master
    f. Import ecen5823-assignment5-<username> into Simplicity Studio
        i. Remember to perform a **Clean…** immediately after you import the project into Simplicity Studio.

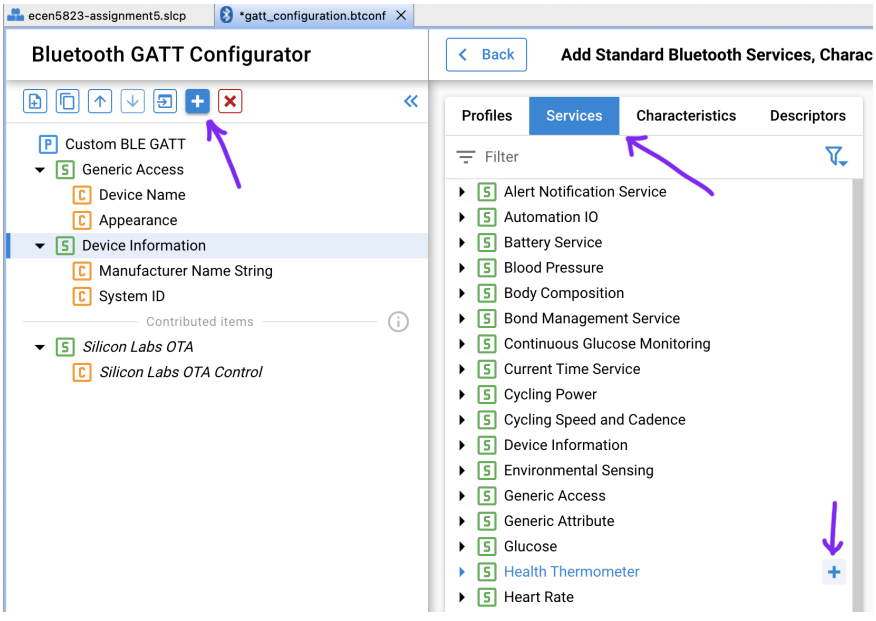
2. You will now be allowing the Bluetooth Stack to manage events, so you will need to delete (comment out) your code in app_process_action() in app.c, it should be empty for A5 through A9. Refer to Lecture 10 for details.
3. Download and install the EFR Connect app to your iOS or Android phone.
4. Now you are ready to begin putting your first Bluetooth Service together
    a. Add the Bluetooth Health Thermometer Service to your application. Open the .slcp file, then select CONFIGURATION TOOLS near the top of the window. Then open the Bluetooth GATT Configurator. You should see the base GATT Database (DB) displayed on the left:



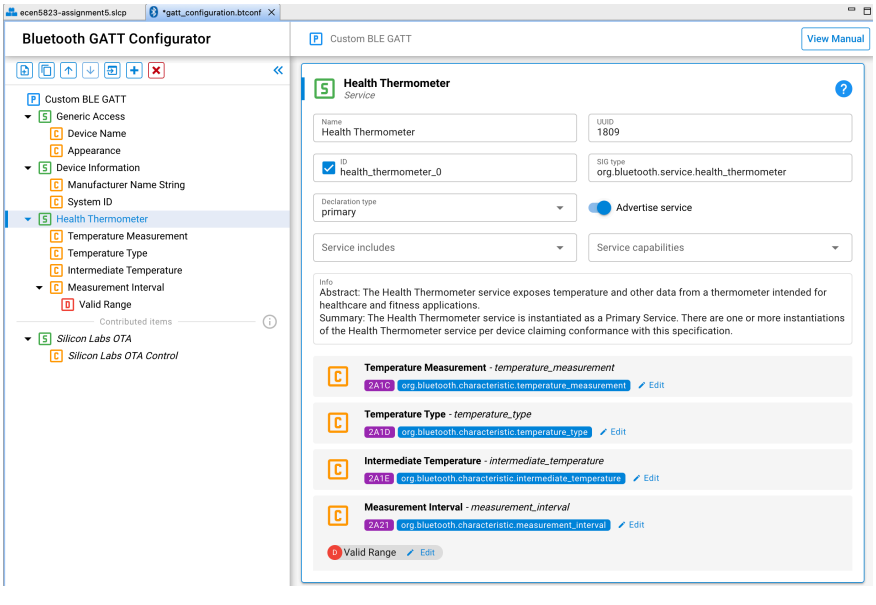
    b. Click on the bigger “+” icon to add a standard Bluetooth GATT item.



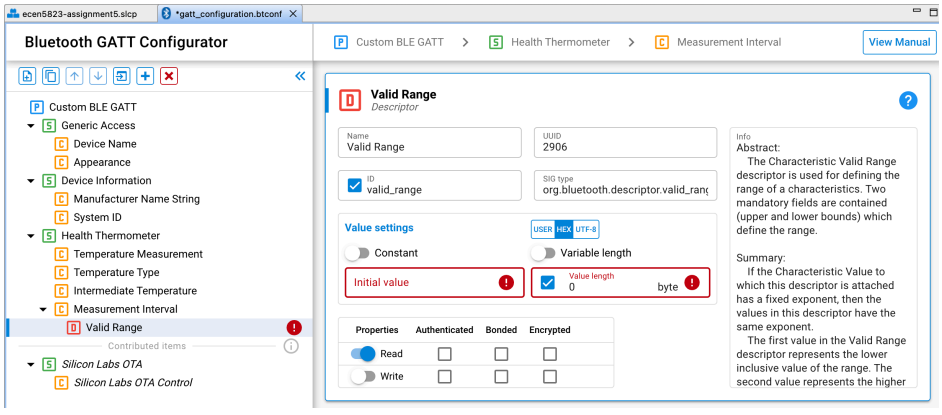
c. At the top of the Add Standard Bluetooth Services... page, click on Services. Then scroll down to Health Thermometer and click the “+” on Health Thermometer to add that Service to your GATT DB.

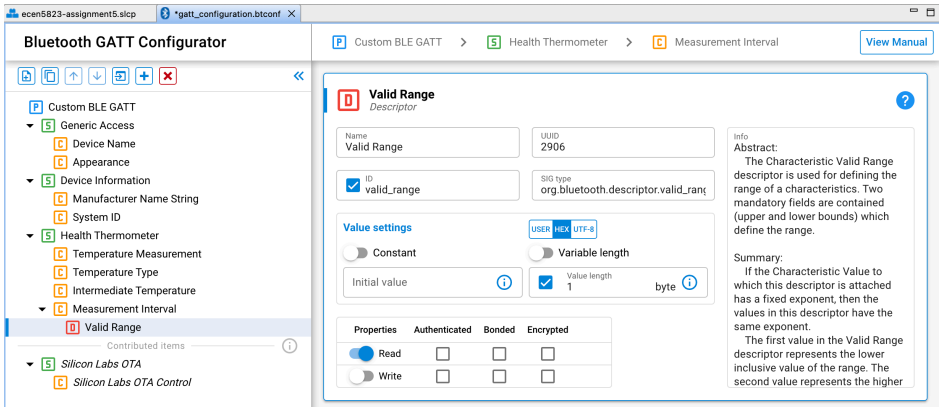


d. Click on the newly added Health Thermometer Service and then click on “Advertise Service” button.

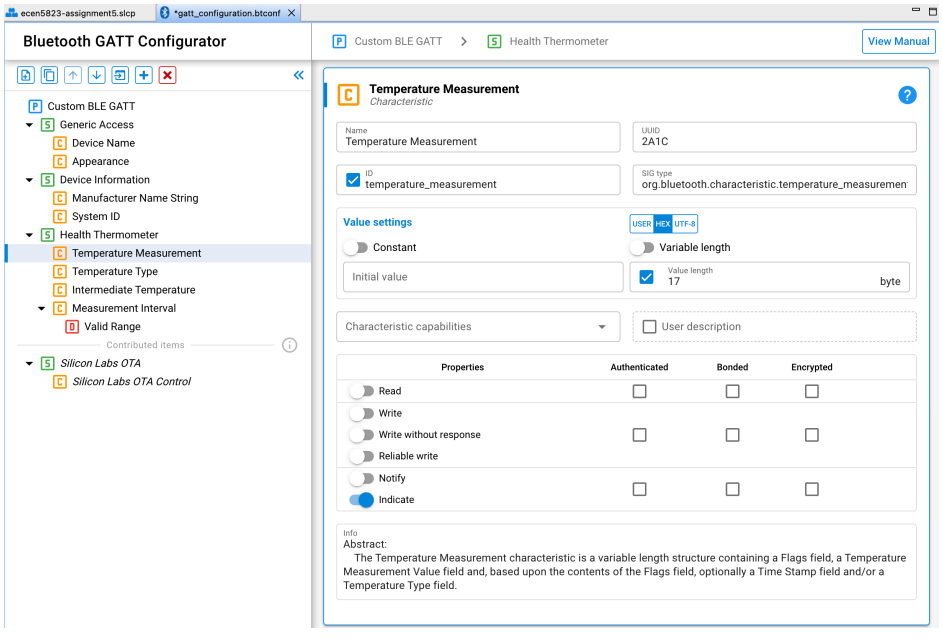


e. You will see an error for the Valid Range Descriptor, Value length field. You will need to correct that by editing the length to be the value of 1.

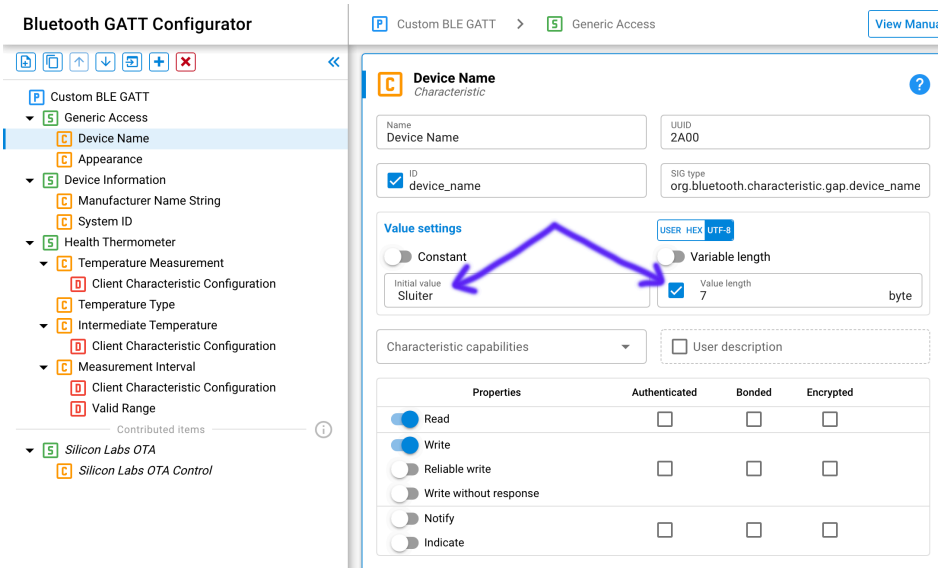




f. Verify that indications are enabled for the Temperature Measurement Characteristic. It should look like this:



g. To make it easier for you to find your device when using the EFR Connect app, set your device name to your name; select either your last name or first name, it's your choice.



h. Save your gatt_configuration.btconf file. This should trigger the "generator' to run and update autogen/gatt_db.c/.h files. In the project explorer select the gatt_db.c file and examine its properties. The file should show it was just modified. This is how you can tell that the generator has run. If for some reason the generator doesn't run, there is a Force Generation button on the Overview tab when you open the .slcp file.

i. Create 2 new files to hold your BLE code, src/ble.c/.h. In ble.h define the following:

```
#define UINT8_TO_BITSTREAM(p, n)       { *(p)++ = (uint8_t)(n); }

#define UINT32_TO_BITSTREAM(p, n)      { *(p)++ = (uint8_t)(n); *(p)++ = (uint8_t)((n) >> 8); \
                                         *(p)++ = (uint8_t)((n) >> 16); *(p)++ = (uint8_t)((n) >> 24); }

#define INT32_TO_FLOAT(m, e)          ( (int32_t) (((uint32_t) m) & 0x00FFFFFU) | (((uint32_t) e) << 24) )


// BLE Data Structure, save all of our private BT data in here.
// Modern C (circa 2021 does it this way)
// typedef ble_data_struct_t is referred to as an anonymous struct definition
typedef struct {

  // values that are common to servers and clients
  bd_addr       myAddress;
 …

  // values unique for server
  // The advertising set handle allocated from Bluetooth stack.
  uint8_t       advertisingSetHandle;
 …


  // values unique for client


} ble_data_struct_t;
```

Then in ble.c declare the storage for your BLE data:

```
// BLE private data
ble_data_struct_t      ble_data;
```

And create a function in ble.c that returns a pointer to a type of ble_data_struct_t. When other .c files need access to the data in ble_data_struct_t, you can call this function to return a pointer to that data structure. Remember: Use of the C keyword “extern” is not allowed in this course.

j. Your application should update the Health Temperature characteristic value in the GATT DB when your previous assignment completes a temperature measurement, on 3 second intervals, just like A3 and A4, and send an indication to the Client if HTM indications are enabled, on 3 second intervals. Macro INT32_TO_FLOAT() may be useful for this purpose. **Important**: Your Server shall only send Health Temperature characteristic indications when there is an open connection. It’s a good idea to implement a bool variable that tracks the open-closed status of the connection. Remember, indications for each characteristic value can be independently enabled/disabled by a Client. It would be a good idea to create a bool variable for each characteristic that then can be used to track whether indications have been enabled for each individual characteristic, and implement an if (variable) { sl_bt_gatt_server_send_indication(); } structure.

5. To maximize energy savings, the Bluetooth Server application should change its advertising, connection interval, and slave latency to what is appropriate to the application. For this assignment:

a. Set the Advertising minimum and maximum to 250mS
b. Set Connection Interval minimum and maximum to 75mS
c. Set the Slave latency to enable it to be “off the air” for up to 300mS
d. Set the Supervision timeout to a value greater than (1 + slave latency) * (connection_interval * 2). See API documentation.
    i. **Important**: When a slave requests a set of connection parameters, they are sent to the master. The master examines the slave’s connection parameters and then sets the connection parameters how the master wants and the slave sets the same values. API calls like sl_bt_connection_set_parameters() will generate a sl_bt_evt_connection_parameters_id event that indicates how the master has set the connection parameters. Include this event in your event handler and LOG the values of interval, latency and timeout, taking into account the units these values are expressed in, in the data structure returned with this event. Take these values as set by the master into account when answering the questions for this assignment. Once you’ve answered the questions, comment out or #define out these out LOG() calls once you have observed the LOG’ed values.

6. Place all of your BT stack event handling code, handle_ble_event() or whatever you call your function, and any other BLE event handling code in ble.c/.h. Edit app_process_action() and comment out all code in that function. Then in the function sl_bt_on_event() edit your code to look like:

```
void sl_bt_on_event(sl_bt_msg_t *evt)
{

  // Some events require responses from our application code,
  // and don't necessarily advance our state machines.
  // For assignment 5 uncomment the next 2 function calls
  handle_ble_event(evt); // put this code in ble.c/.h

  // Sequence through states driven by events.
  // Modify your state machine to look for the appropriate event ID value,
  // and extsignals value you passed from your schedulerSetEventXXX() functions.
  state_machine(evt);    // put this code in scheduler.c/.h

} // sl_bt_on_event()
```

7. In your app_init() function, limit the lowest energy mode to EM2. If you allow the MCU to sleep to EM3, it will never wake up. So you’ll put this requirement in place and never remove it.
8. Build a state machine to control the sequencing of taking a temperature measurement where the state machine is driven by events. Put that code in scheduler.c/.h.
9. For the code you submit for grading, there should be no calls to __WFI() or __BKPT() or LOG_***() functions. No calls to Systick routines are allowed. However, do log errors returned from API calls. For a fully functioning program there should be no errors returned from API calls, therefore there should be no LOG messages in the terminal.

10. Use the EFR Connect phone app to verify you can receive temperature indications for the attribute.
11. Bonus Opportunity: You can earn 10% extra credit in this assignment if your implementation only takes a temperature measurement at 3 second intervals **while a BLE connection is open** and **indications for the Health Thermometer temperature are enabled**. Additionally for the +10 extra credit, your program will stop taking temperature measurements if the BLE connection is disconnected, or HTM indications are disabled.
12. Things to think about:
    a. What should the states in your state machine do if the Bluetooth connection closes before it completes a temperature measurement sequence?
    b. It is a very good design practice to maintain a variable that tracks whether your Server has an indication in-flight or not. Implement this now, you'll thank me later.
    c. Your state machine from A3 and A4 accepted a uint32_t value (typically in most students implementations). Your state machine is now receiving a pointer to a BT stack data structure (that union struct we discussed in class) for every BT stack event. You will have to modify your state machine to look for:
        i. The appropriate event ID, and
        ii. The signal value sent from your schedulerSetEvenetXXX() function in the event structure field: data.evt_system_external_signal.extsignals. Remember: The extsignals field can contain multiple bits set!

Questions:
Answers to the Assignment5-HealthThermometerProfile-BLE.md file in the questions folder, included in your submission.

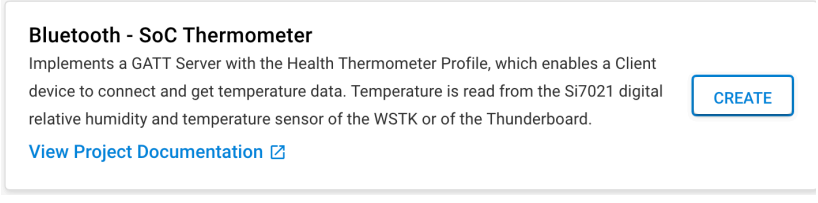
Deliverables:

- Submission via **github classroom** and your repository setup via the classroom link at above. Use the following git command to submit your assignment:
  git push origin master
- Verify your ecen5823-assignment5-<username> repository contains your latest/intended code and answers to the questions. It is your responsibility to push the correct version of your project/code. Use a web browser to verify the correct version of your project/code has been uploaded to GitHub.
- In **Canvas**, submit the URL to your github repository and in **GitHub** create a tag of your GitHub submission prior to the due date with the text of A5-final. The date and time of the tag creation is what we use to determine whether your submission was on time or late. These 2 steps will complete your assignment submission.

Approach/Guidance

Please see the spreadsheet "Assignment5-BLEEventSchedulerGuidance.xlsx or .numbers" in the questions/ folder in the assignment project. See also the document: Assignment 5 - Guide to Bluetooth Application Development.pdf in the A5 assignment in Canvas.

There is a sample project called Bluetooth - SoC Thermometer. This could be a great reference. Look for the function which takes the temperature reading and sends out that temperature value in an indication in IEEE-11073 32-bit floating point format to the Phone app using sl_bt_gatt_server_send_indication(). This project can be found in the Launcher view of Simplicity studio:



Please don't structure your code the way you see it in the Silicon Labs example. Please structure your code following the guidance I've provided in class. The thing you can learn from this example is: for a given event, what API calls should I make in response to that event?

The following articles may help you come up to speed on the SiLabs BLE API:

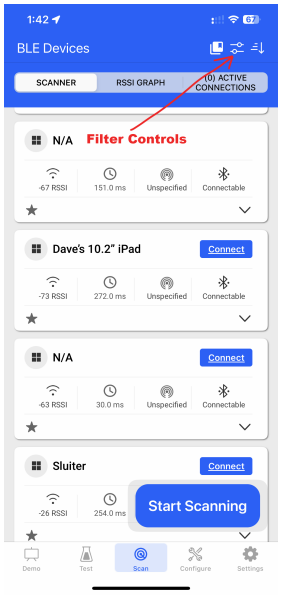
https://www.silabs.com/documents/public/user-guides/ug136-ble-c-soc-dev-guide.pdf

https://docs.silabs.com/bluetooth/latest/general/gatt-protocol/gatt-server-and-client-roles

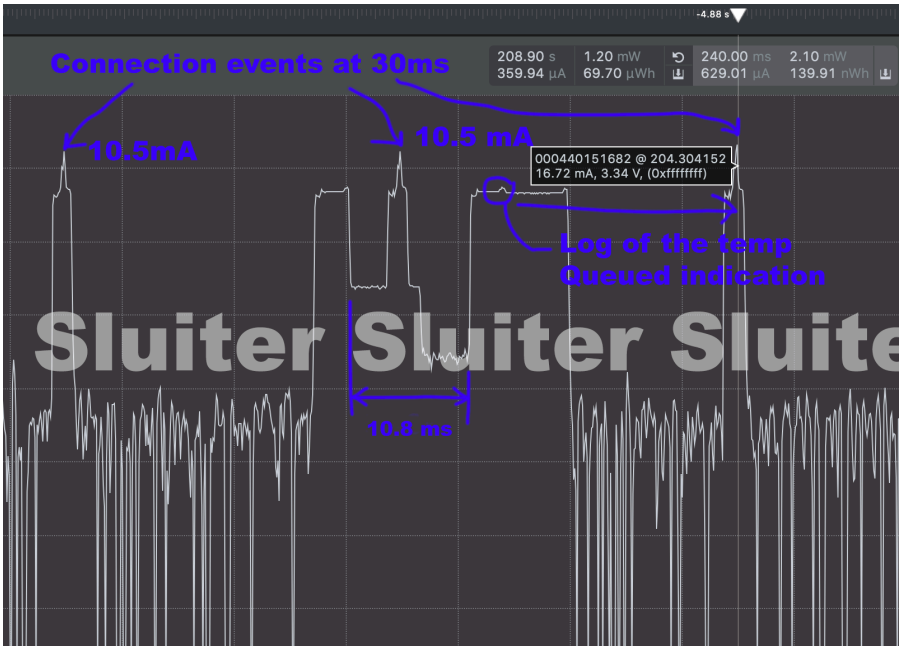
At a minimum your program will need to handle the following BT stack events:

```
sl_bt_evt_system_boot_id
sl_bt_evt_connection_opened_id
sl_bt_evt_connection_closed_id
sl_bt_evt_connection_parameters_id
sl_bt_evt_system_external_signal_id
sl_bt_evt_gatt_server_characteristic_status_id
sl_bt_evt_gatt_server_indication_timeout_id
```

Screen capture of EFR Connect App showing your Server and advertising its Device Name. Note the filter controls at the top to help you find your Gecko (I marked that in red).



Students often ask me what the current profile should look like. At the time of this writing and corresponding version of the EFR Connect phone app, this is what I see:



This is an interesting one. Note that there is a connection event right in the middle of when the 7021 is taking a temp measurement. Note when the indication is queued (sent as far as the Application layer is concerned), but it isn't actually transmitted to the Client until the next connection event, see that the peak current is higher on that 3rd connection event in the pic.

Files in your src/ directory should look like: