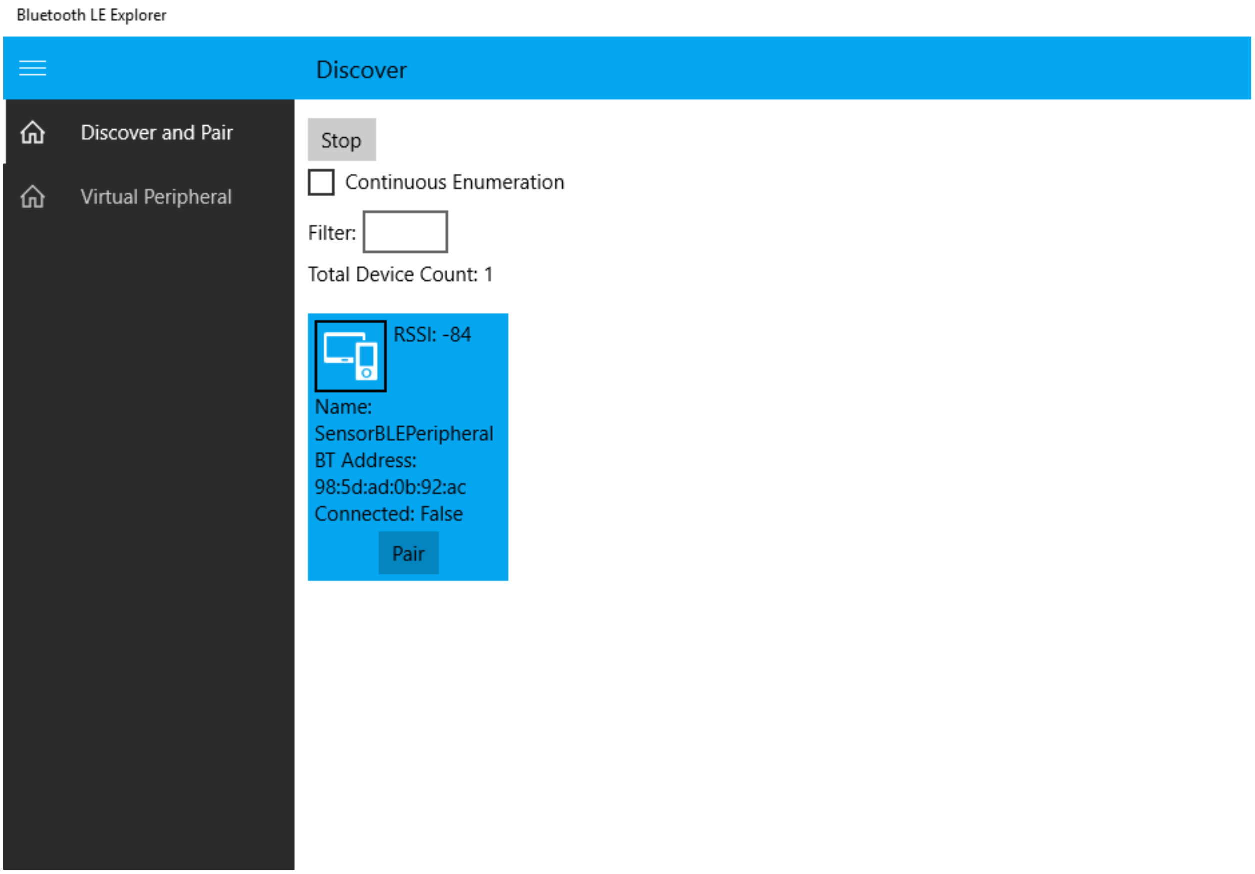
CC2541 BLE transmission of data

This short introduction will explain how the cc2541 circuitry is sending data using BLE



☐ Continuous Enumeration

Filter:

Total Device Count: 1

RSSI: -84

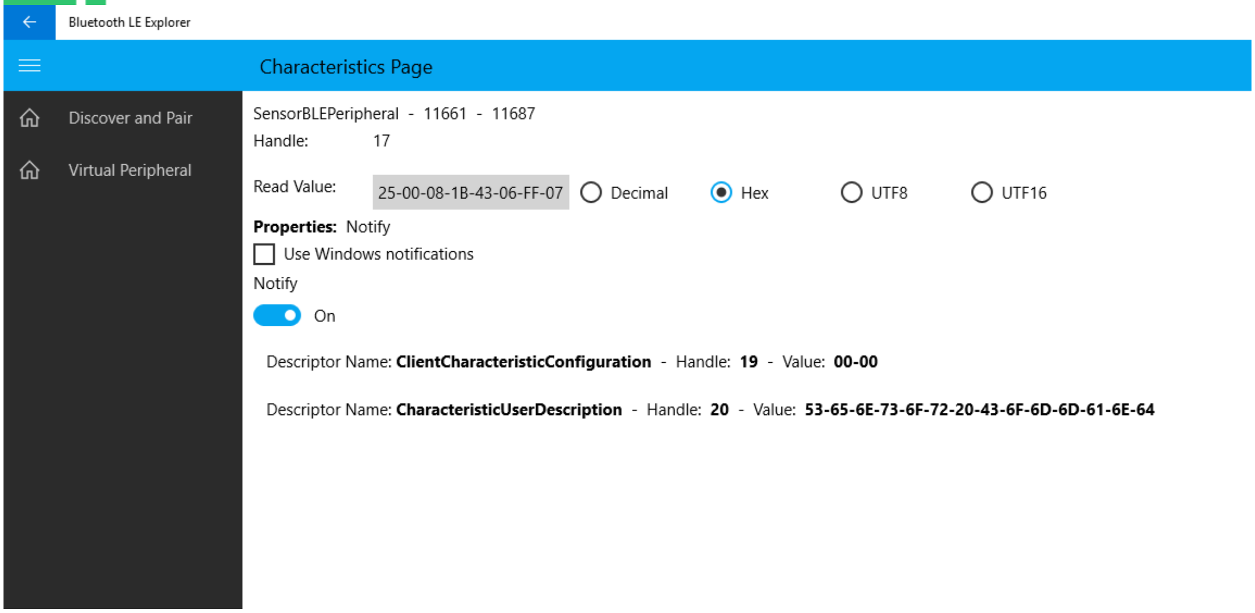
Name: SensorBLEPeripheral

BT Address: 98:5d:ad:0b:92:ac

Connected: False

Connect to device by just clicking the blue box (Do not attempt to pair the device as this will not work)

When connected to the SensorBLEPeripheral a Device Services Page will appear containing all the information that the cc2541 is currently transmitting. The GenericAcess and GenericAttribute services provides general information about the device. The service for BLE communication is service 11661 which can be found further down the list. The first handle in this service is in this case handle 17 with the user description “sensor command”. This value gives a read not permitted initially because the value to communicate it must be set to “Notify”. Clicking this handle and enabling notify will make the value update whenever it changes allowing a readout of values.



SensorBLEPeripheral - 11661 - 11687

Handle: 17

Read Value: 25-00-08-1B-43-06-FF-07 ○ Decimal ● Hex ○ UTF8 ○ UTF16

Properties: Notify

□ Use Windows notifications

Notify

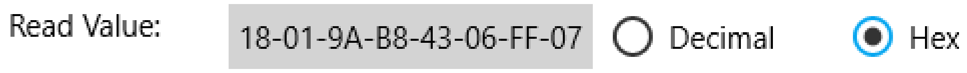
On

Descriptor Name: **ClientCharacteristicConfiguration** - Handle: **19** - Value: **00-00**

Descriptor Name: **CharacteristicUserDescription** - Handle: **20** - Value: **53-65-6E-73-6F-72-20-43-6F-6D-6D-61-6E-64**

Interpreting the 16 char hex value

The observed data will look the following



Read Value: 18-01-9A-B8-43-06-FF-07 ○ Decimal ● Hex

This hex is separated into counter, time, temperature and ADC_value

Each parameter has 4 bytes, but they are divided into two section. This means that for this string

0118 = Counter → 280 decimal value

B89A = time → 47258 decimal value

0643 = Temperature →converted to decimal gives 1603 which can be converted to 160.3 Celsius(Example code was not set thus sensor readout will show 160.3 Degree celcius)

07FF = ADC_Value → 2047 decimal → using the formula explained later the conversion from the 2047 adc value gives 2.998535 V.

Temperature value is the measured value divided by 10.

The adc value can be converted by using the values shown on handle 21 and 24 named denominator D0 and Numerator N0. For this reading D0 is 204800 and N0 is 300.

Vout can then be calculated with the following formula

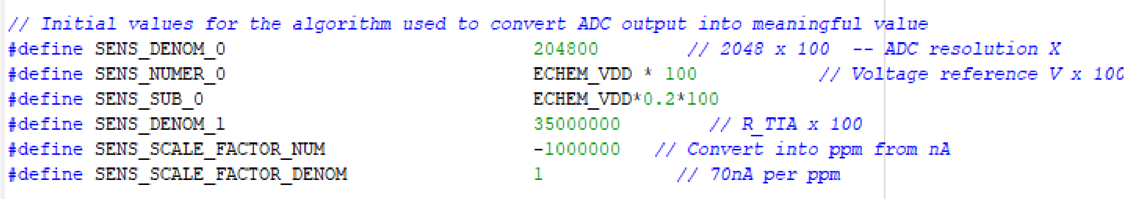
$$Vout = \left(\frac{adc\_value}{D0}\right) * N0$$

Formula for converting Vout to current using the received parameters from cc2541.

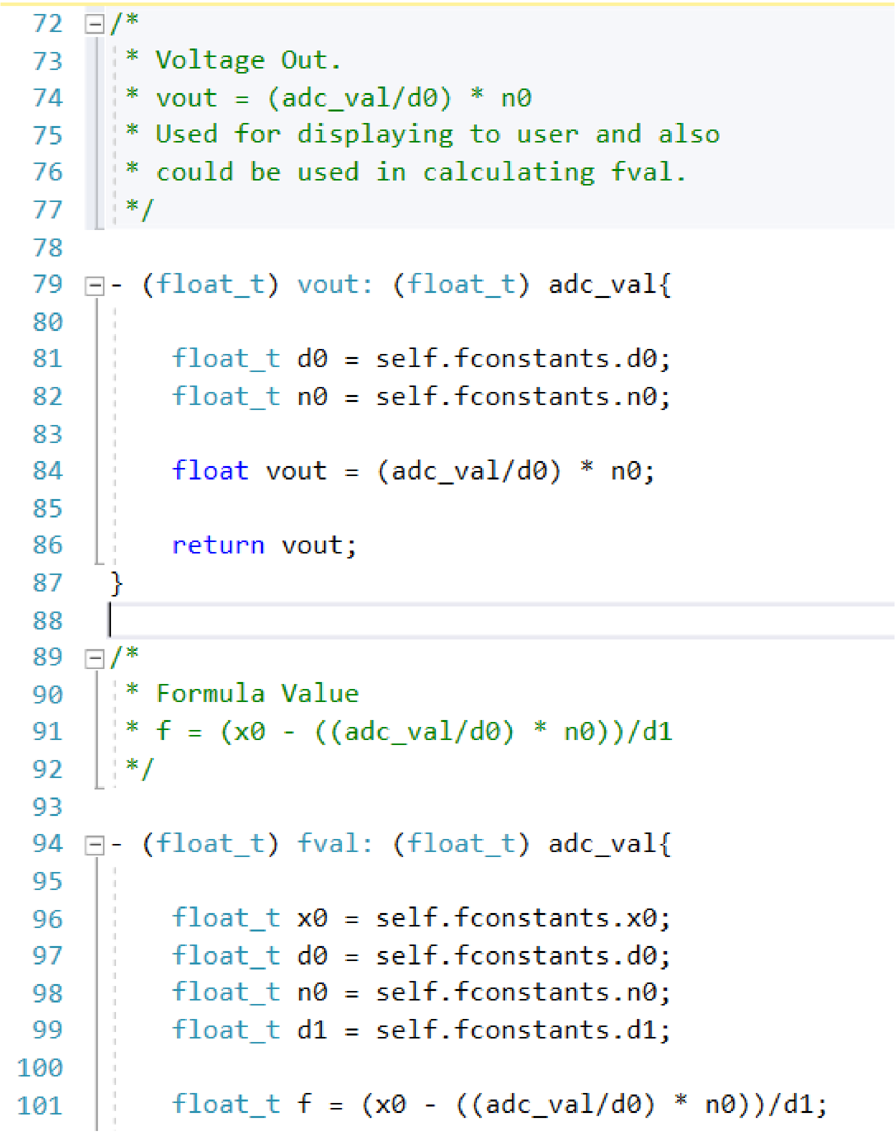
$$Current = (x0 - \frac{\left(\left(\frac{adc\_value}{D0}\right) * N0\right)}{D1}$$

X0 and D1 should be able to be viewed on the handles after D0 and N0. These values are set in the firmware code of the cc2541.

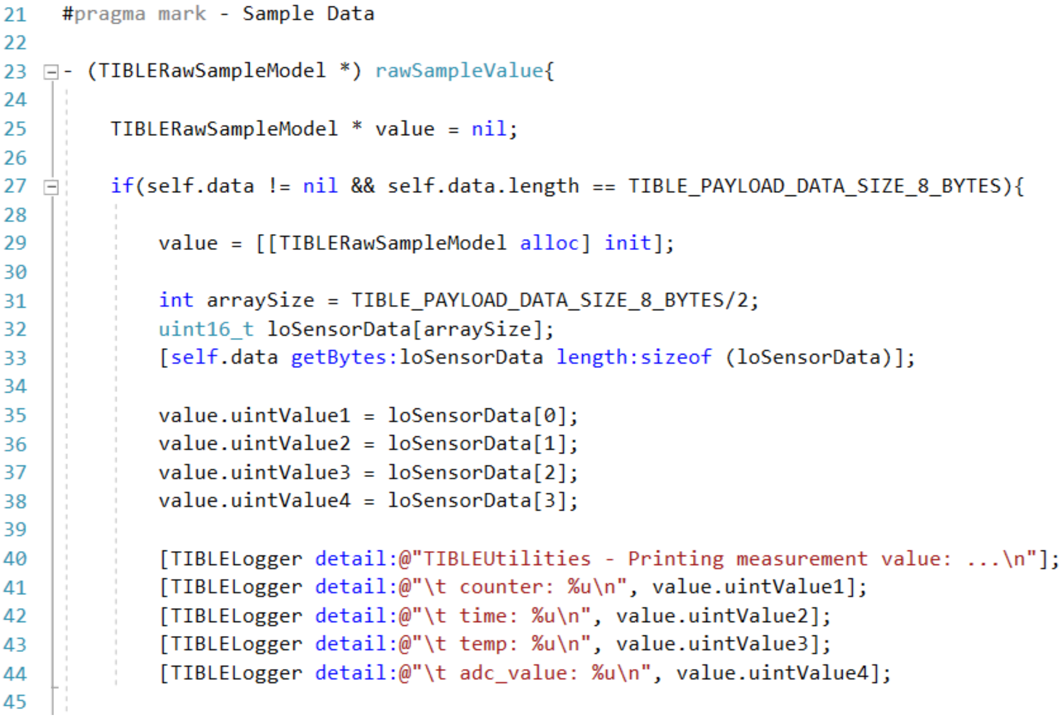
Snippets from sourcecode for reference



```
// Initial values for the algorithm used to convert ADC output into meaningful value
#define SENS_DENOM_0                          204800        // 2048 x 100  -- ADC resolution X
#define SENS_NUMER_0                          ECHEM_VDD * 100            // Voltage reference V x 100
#define SENS_SUB_0                            ECHEM_VDD*0.2*100
#define SENS_DENOM_1                          35000000        // R_TIA x 100
#define SENS_SCALE_FACTOR_NUM                 -1000000    // Convert into ppm from nA
#define SENS_SCALE_FACTOR_DENOM               1             // 70nA per ppm
```



```
/*
 * Voltage Out.
 * vout = (adc_val/d0) * n0
 * Used for displaying to user and also
 * could be used in calculating fval.
 */

- (float_t) vout: (float_t) adc_val{

    float_t d0 = self.fconstants.d0;
    float_t n0 = self.fconstants.n0;

    float vout = (adc_val/d0) * n0;

    return vout;
}

/*
 * Formula Value
 * f = (x0 - ((adc_val/d0) * n0))/d1
 */

- (float_t) fval: (float_t) adc_val{

    float_t x0 = self.fconstants.x0;
    float_t d0 = self.fconstants.d0;
    float_t n0 = self.fconstants.n0;
    float_t d1 = self.fconstants.d1;

    float_t f = (x0 - ((adc_val/d0) * n0))/d1;
```



```
#pragma mark - Sample Data

- (TIBLERawSampleModel *) rawSampleValue{

    TIBLERawSampleModel * value = nil;

    if(self.data != nil && self.data.length == TIBLE_PAYLOAD_DATA_SIZE_8_BYTES){

        value = [[TIBLERawSampleModel alloc] init];

        int arraySize = TIBLE_PAYLOAD_DATA_SIZE_8_BYTES/2;
        uint16_t loSensorData[arraySize];
        [self.data getBytes:loSensorData length:sizeof (loSensorData)];

        value.uintValue1 = loSensorData[0];
        value.uintValue2 = loSensorData[1];
        value.uintValue3 = loSensorData[2];
        value.uintValue4 = loSensorData[3];

        [TIBLELogger detail:@"TIBLEUtilities - Printing measurement value: ...\n"];
        [TIBLELogger detail:@"\t counter: %u\n", value.uintValue1];
        [TIBLELogger detail:@"\t time: %u\n", value.uintValue2];
        [TIBLELogger detail:@"\t temp: %u\n", value.uintValue3];
        [TIBLELogger detail:@"\t adc_value: %u\n", value.uintValue4];

```