MAKING SENSE OF BLUETOOTH BEACON DATA

1. Introduction

The ORB has a Bluetooth peripheral that can listen for Bluetooth Low Energy (BLE) advertising packets. BLE beacons typically use the advertising packets to communicate measured data such as temperatures, voltages, movement, and battery voltage. BLE tags are a special type of beacon that typically only contain identification information and are used to locate items.

BLE beacons send advertising messages at different rates. Some report every second and some may be every minute or more. Battery operated BLE devices tend to send at lower rates to save power. Some BLE device are smart and will slow their send rate if they are not being used. A tire pressure monitoring device may stop sending if the tire is not rotating.

Typical protocols used in advertising packets include Eddystone and iBeacon. The ORB supports both and more. Although standards exist, the format of the data sent by BLE beacons is generally set by the manufacturer. Specifications in the datasheet will inform how to interpret the data portion of the BLE packet. This application note looks at extracting temperature and id from a temperature and id sensor from ELA.

Figure 1 - ELA Temperature Beacon

2. Capturing BLE Messages

When active, the Bluetooth peripheral on the ORB will scan for all advertising packets. In a typical environment, phones, computers, and other devices are all advertising. The ORB Bluetooth peripheral could easily report a dozen BLE devices even when the one you are searching for is off. The ORB can filter for only the required BLE devices by filling in the *Address Capture List*. For further details on configuring the BLE peripheral, please refer to the ORB User Guide: http://docs.senquip.com/orbug/.

This application note assumes that the ELA temperature and id sensor has been turned on and is sending advertising packets. To assist in the activation and setup of BLE beacons, ELA have an app known as "Device Manager Mobile". This application note does not cover use of this application.

To identify tags and beacons and to identify their identifiers, we will use a "BLE Scanner" application called from Bluepixel Technologies. The app can be downloaded from the Android app store.

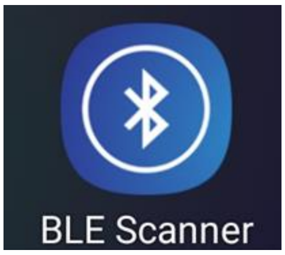
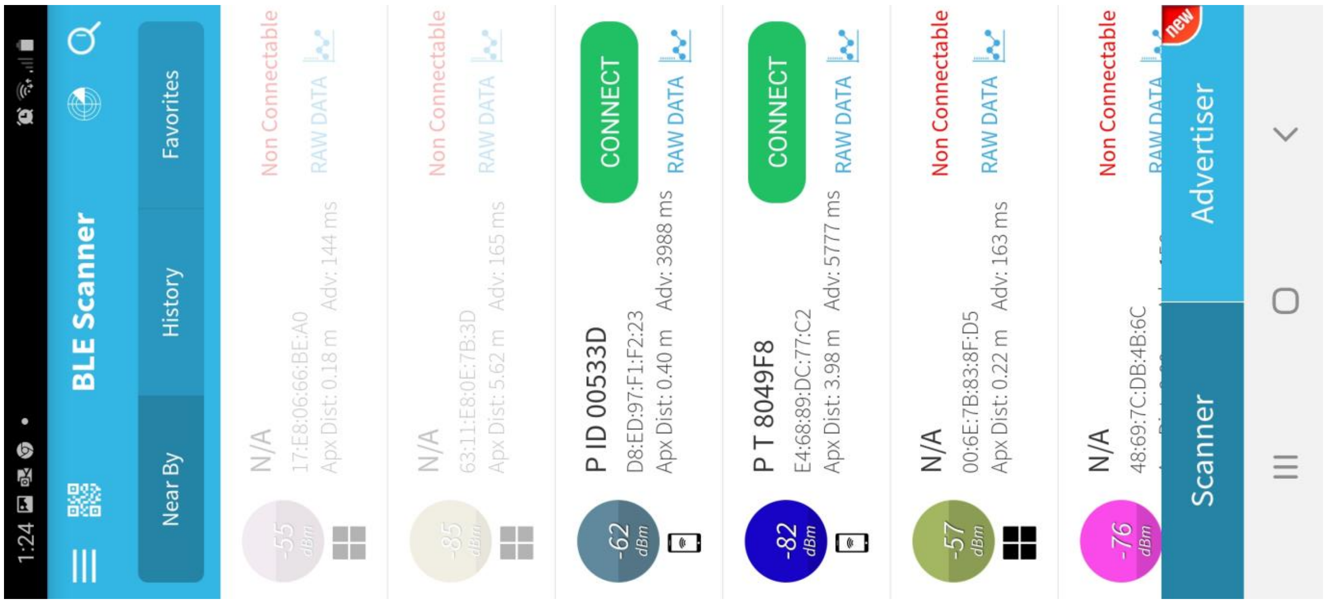


Figure 2 - BLE Scanner app from Bluepixel Technologies

Using the app, as shown in Figure 3, we see that there are various devices nearby. The two that we are looking for are "P ID 00533D " (hex id d8ed97f1f223) and "P T 8049F8" (hex id e46889dc77c2).

Identifier (hex)	Beacon Type	Name Printed on Label
d8ed97f1f223	Identifier only	P ID 00533D
e46889dc77c2	Temperature and identifier	P T 8049F8

The hex identifiers will be used in the ORB Bluetooth setup, as shown in Figure 4, to restrict the number of BLE packets captured. In this case, we know those are the identifiers we are looking for because we noted it when activating the devices with the ELA mobile app and the name matches that printed on the sensor. It should be noted that the numbers printed on the devices are not always the same as the identifiers. If you are unsure of the identifiers you are searching for, take the BLE devices to an area far from electronic noise and do a scan.


1:24

BLE Scanner

Near By | History | Favorites

-55 dBm N/A 17:E8:06:66:BE:A0 Apx Dist: 0.18 m Adv: 144 ms Non Connectable RAW DATA

-85 dBm N/A 63:11:E8:0E:7B:3D Apx Dist: 5.62 m Adv: 165 ms Non Connectable RAW DATA

-62 dBm P ID 00533D D8:ED:97:F1:F2:23 Apx Dist: 0.40 m Adv: 3988 ms CONNECT RAW DATA

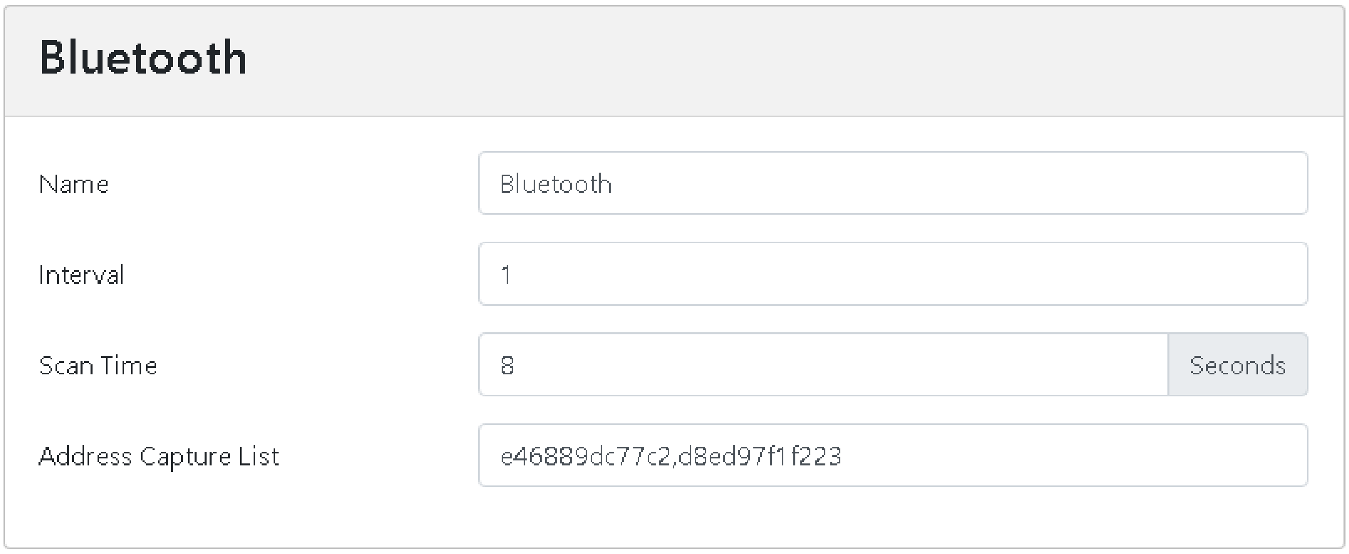
-82 dBm P T 8049F8 E4:68:89:DC:77:C2 Apx Dist: 3.98 m Adv: 5777 ms CONNECT RAW DATA

-57 dBm N/A 00:6E:7B:83:8F:D5 Apx Dist: 0.22 m Adv: 163 ms Non Connectable RAW DATA

-76 dBm N/A 48:69:7C:DB:4B:6C Non Connectable RAW DATA

Scanner | Advertiser new

Figure 3 - BLE Scan Results



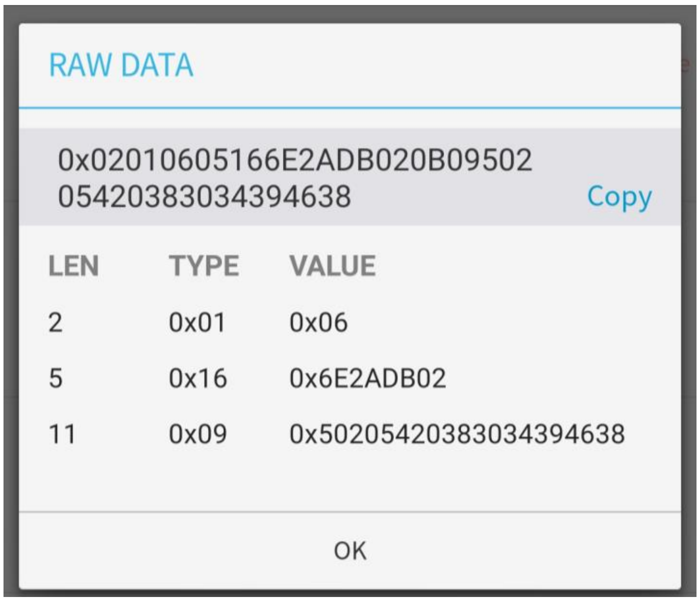
Bluetooth

Name	Bluetooth
Interval	1
Scan Time	8 Seconds
Address Capture List	e46889dc77c2,d8ed97f1f223

Figure 4 - Restricting the captured BLE Packets

By pressing the RAW DATA button in the BLE Scan app, as shown in Figure 3, we can see details of the data received from a particular beacon. In this case, the raw data from the temperature and id beacon consists of a hex number:

02010605166E2ADB020B0950205420383034394638.



RAW DATA

0x02010605166E2ADB020B09502
05420383034394638

LEN	TYPE	VALUE
2	0x01	0x06
5	0x16	0x6E2ADB02
11	0x09	0x5020542038303434394638

Figure 5 - Raw Data from BLE Temperature Beacon

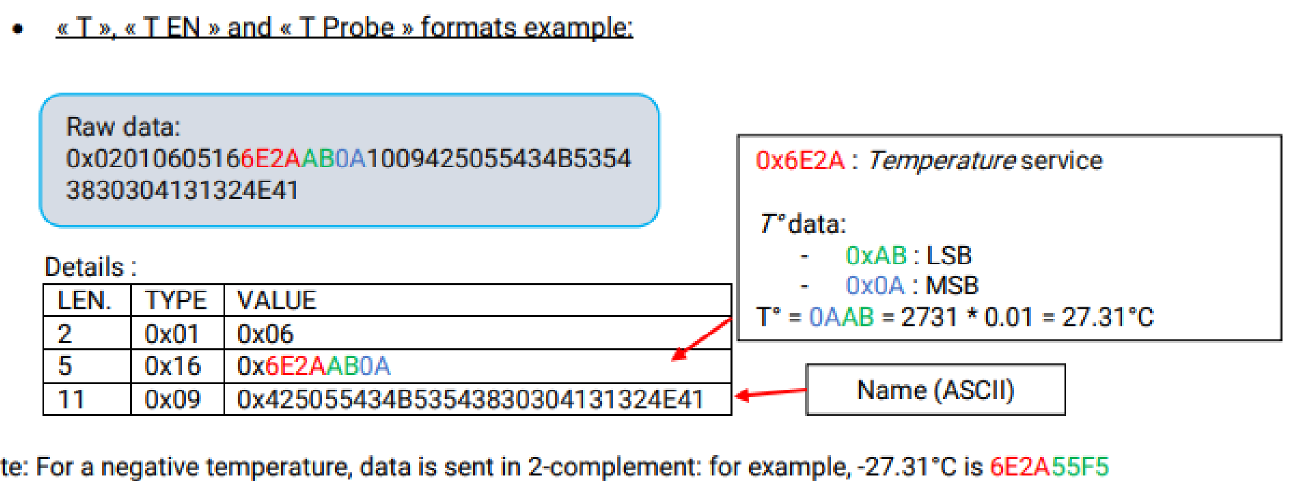
The hex number contains all the data transmitted in the BLE device advertising message. The message contains various types of information of different lengths. The types of information are defined in the Generic Access Profile (GAP) and are described here: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/.

In the case of our message from the temperature and id beacon, we have the following data:

Length	Type	Type Description	Data	Manufacturer Description
2	0x01	Flags	0x06	
5	0x16	Service Data	6E2ADB02	Data field containing temperature
11	0x09	Complete Local Name	50205420383034394638	Device name

Table 1 - Beacon Data

Looking at the manufacturer datasheet shown in Figure 6, we see the description of the data field lines up with the expectation in Table 1. Note that the data from the id only beacon will not have the Service Data field as it does not have any data to transmit.



- « T », « T EN » and « T Probe » formats example:

Raw data:
0x02010605166E2AAB0A1009425055434B5354
3830304131324E41

Details :

LEN.	TYPE	VALUE
2	0x01	0x06
5	0x16	0x6E2AAB0A
11	0x09	0x425055434B53543830304131324E41

0x6E2A : *Temperature* service

T° data:
- 0xAB : LSB
- 0x0A : MSB

T° = 0AAB = 2731 * 0.01 = 27.31°C

Name (ASCII)

Note: For a negative temperature, data is sent in 2-complement: for example, -27.31°C is 6E2A55F5

Figure 6 - Manufacturer Data Description

2.1. Parsing the Temperature Data

In the example above, the service data is: 6E2ADB02

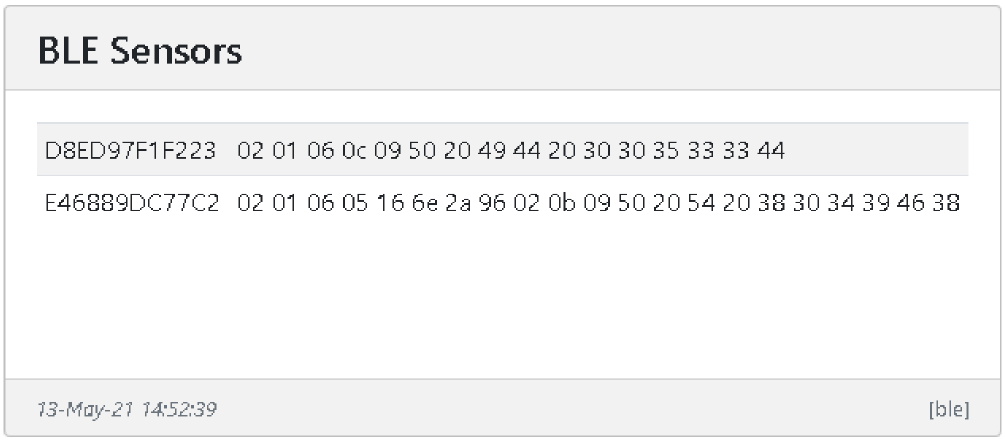
From the datasheet, the Service Data contains a 2-byte data type indicator, in this case 6E2A which indicates temperature data. The temperature data is contained in the next 2 bytes in Intel byte order with the most significant byte last (AB0A). Reversing the byte order results in 0AAB, which in decimal is 2731. As per the datasheet, we scale the result by multiplying with 0.01 to realise a temperature of 27.31 degrees in Celsius.

2.2. Reading the Device Name

In the example, the Complete Local Name data is: 50205420383034394638

From the datasheet, the data is in ASCII in hex number format. Converting to text results in "P T 8049F8" which is the name printed on the device.

A scan from the BLE peripheral delivers the data shown in Figure 7 on the Senquip Portal. Notice that the first device, which is the id only beacon does not have the Service Data field as it does not have any temperature data to transmit.



BLE Sensors

```
D8ED97F1F223   02 01 06 0c 09 50 20 49 44 20 30 30 35 33 33 44
E46889DC77C2   02 01 06 05 16 6e 2a 96 02 0b 09 50 20 54 20 38 30 34 39 46 38
```

13-May-21 14:52:39 [ble]

Figure 7 - BLE Widget on Senquip Portal

The ORB also measures received signal strength (RSSI) from each BLE device. The RSSI can be seen in the raw JSON data sent by the ORB. The RSSI can be used to determine if communications with the beacon will be reliable or not. The RSSI being received by the temperature beacon is much lower as it is in a metal fridge for this demo.

```
"ble": [
  {
    "data": "0201060c0950204944203030353333344",
    "rssi": -36,
    "id": "d8ed97f1f223"
  },
  {
    "data": "02010605166e2a96020b09502054203830343946 38",
    "rssi": -67,
    "id": "e46889dc77c2"
  }
```

Figure 8 - Raw JSON Data from ORB

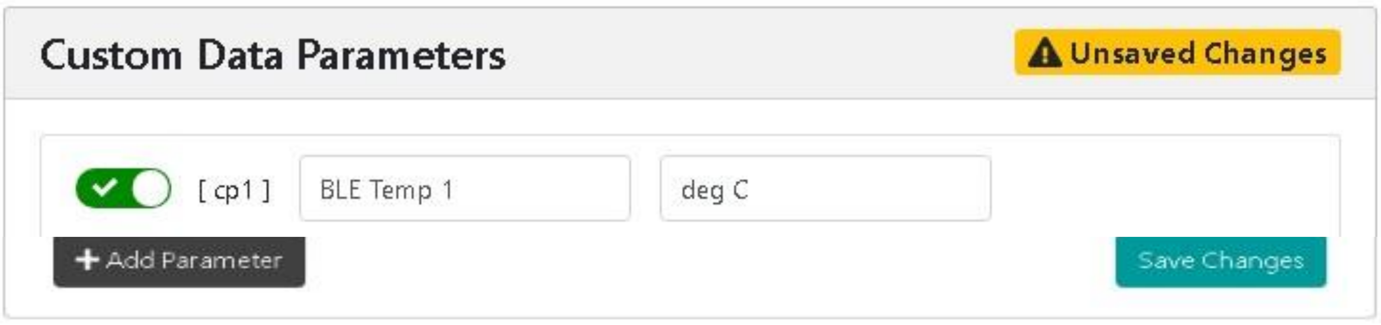
At this point, the BLE data is sent to the Senquip Portal or another server, and the server could parse the data into temperature and id fields. We will however progress to using a JavaScript on the ORB to interpret the data.

3. Parsing the BLE Data with a Script

We will work with the data received from the temperature sensor and will parse the temperature only field.

In this application note, we will assume that you have access to the scripting feature, that the ORB is subscribed to a Premium plan, and that you are familiar with the scripting environment on the Senquip Portal. For further details on scripting on the Senquip ORB, please see the scripting guide: https://docs.senquip.com/scripting_guide.

To be able to monitor the beacon temperature, a custom parameter, BLE Temp 1, will be generated and displayed on the Senquip Portal. Figure 9Figure 9 shows the creation of the custom parameter cp1, BLE Temp 1, in deg C.


Custom Data Parameters

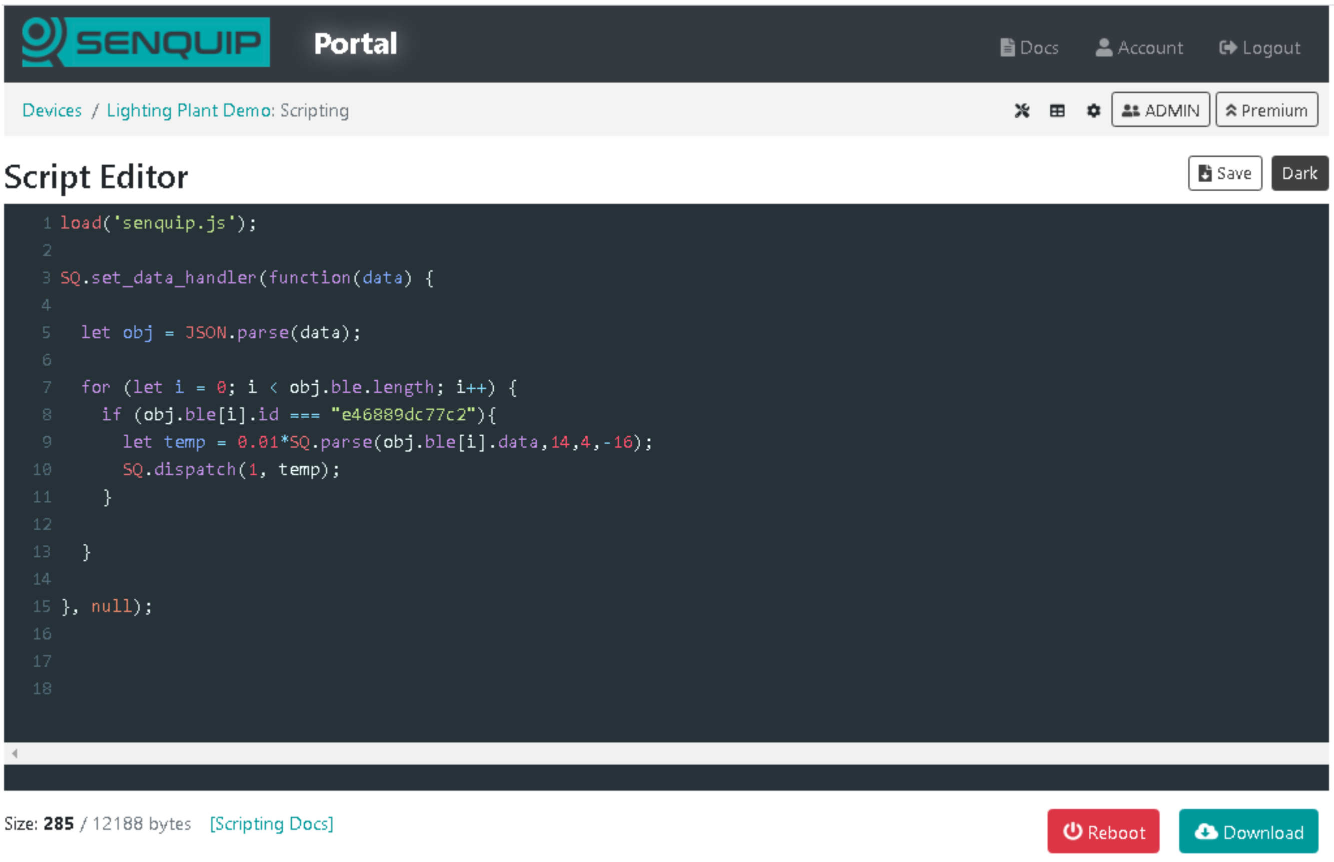
[cp1] BLE Temp 1 — deg C

Figure 9 – A Custom Parameter is Created to Show Beacon Temperature

Figure 10Figure 10 shows the script to parse the BLE beacon message and extract the temperature. A brief explanation for each line is given below. In the example, we will use the following data received from the BLE temperature and id beacon:

02 01 06 05 16 6e 2a 7d 02 0b 09 50 20 54 20 38 30 34 39 46 38

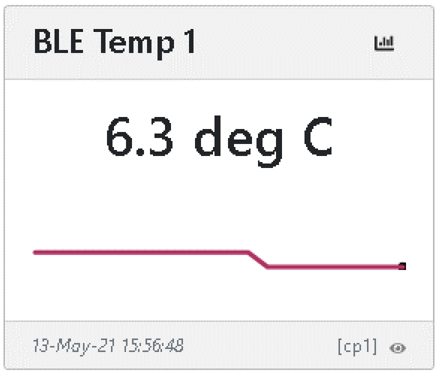
1) Load the Senquip Library that includes all the functions prefixed with SQ
3) Setup the function that will be executed at each base interval, after the measurements have been taken
5) Parse the JSON data to create an object that can be indexed by the JSON key, can1 in this case
7) Cycle through all of the received BLE messages and for each one, extract the required data values
8) Identify the temperature beacon by its identifier (e46889dc77c2)
9) Extract 4 bytes in Intel byte order by starting at character 14 in the message, taking 4 characters (7d02), identifying them as hex and reversing their order (027d = 637). Scale by 0.01 as per the datasheet snippet in Figure 6 to get 6.37 deg C.
10) Dispatch the temperature to the Senquip Portal to be shown as custom parameter 1.
15) Close the function



```
load('senquip.js');

SQ.set_data_handler(function(data) {

  let obj = JSON.parse(data);

  for (let i = 0; i < obj.ble.length; i++) {
    if (obj.ble[i].id === "e46889dc77c2"){
      let temp = 0.01*SQ.parse(obj.ble[i].data,14,4,-16);
      SQ.dispatch(1, temp);
    }

  }

}, null);
```

Figure 10 - Script to parse the electronic engine controller 1 message

Once the script is downloaded and the ORB is rebooted, the function runs, and the portal now shows temperature in deg C.



The code as presented has been written to be simple. It is unnecesarily verbose and an effort should be made to compress the code to conserve memory on the ORB.

The code should be further developed to perform checks on the data to ensure integrity and to protect the JavaScript interpreter from invalid data. The identifier could also be extracted.

4. Conclusions

Users can write their own JavaScript for the Senquip ORB. Scripting allows BLE messages to be parsed to extract various data fields. Those fields can then be converted to numbers, which allows them to be manipulated mathematically, and to create complex exceptions. The calculated values can be dispatched to the Senquip Portal or another server for display and storage.

APPENDIX 1 – SOURCE CODE

```
load('senquip.js');

SQ.set_data_handler(function(data) {

 let obj = JSON.parse(data);

 for (let i = 0; i < obj.ble.length; i++) {
        if (obj.ble[i].id === "e46889dc77c2"){
           let temp = 0.01*SQ.parse(obj.ble[i].data,14,4,-16);
           SQ.dispatch(1, temp);
        }

 }

}, null);
```