Internet of Things

Baby Monitor Design Document

Lecturers: Damon Berry, Michael Core, Frank Duignan, Paula Kelly

Student Name: Jack Harding

Student Number: C15441798

Date: 21 November 2018

Introduction

A baby monitor device is required for a website and must display the status on request of the user, the status includes the direction of the device and acceleration, for the purpose of finding the device, an LED has been fitted, this can be switched on to find the device and off when found. The status of the device is found by entering commands into a MQTT broker, this is connected to the gateway (Raspberry Pi) via NodeJS, the gateway is then connected to the device (BBC Microbit) via Noble.

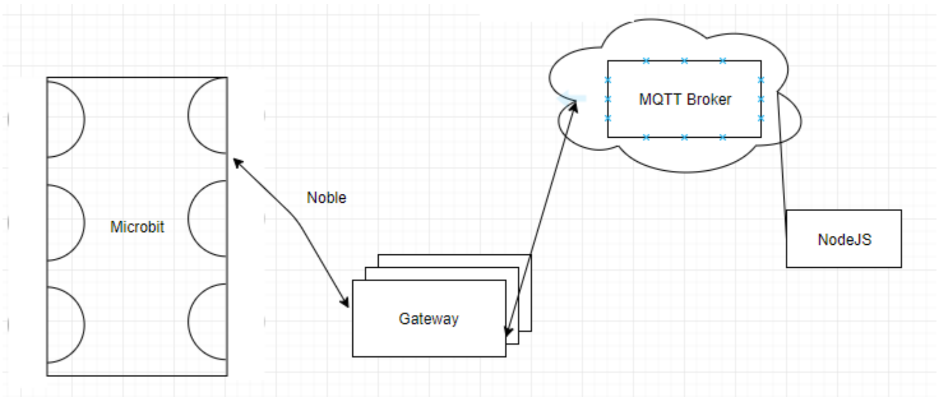


Figure 1 System Architecture

NodeJS is an event driven API written in JavaScript, its asynchronous nature allows updates on the Microbit to be sent as it occurs which is perfect for this application. Noble is an NPM package (Node Package Manager) that allows Bluetooth enabled devices to connect and give Node access to service and characteristics available on the Microbit.

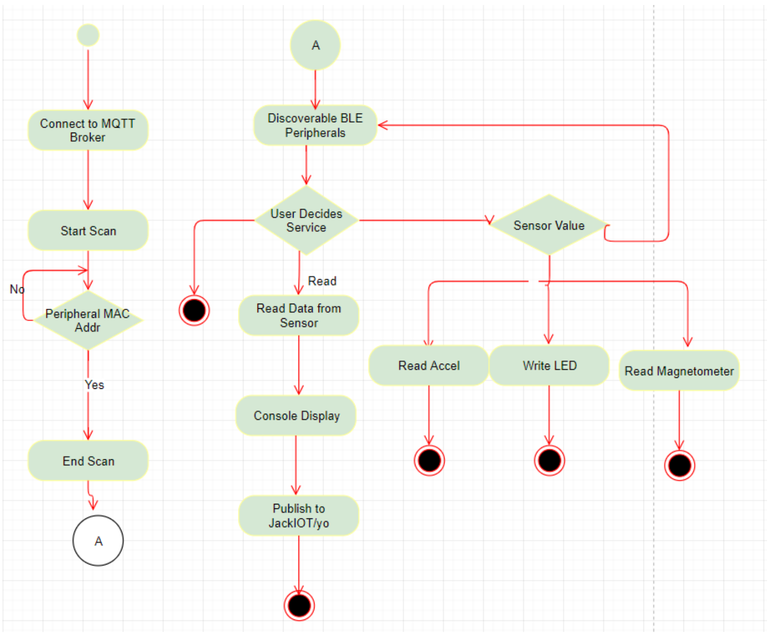


Figure 2 UML Activity Diagram

Figure 2 shows the UML activity diagram, which gives a visual representation of the series of events that occur when using the system.

Process

The variables must be declared to include the necessary functions for the code to operate correctly. The first two are the NPM packages MQTT and Noble, these are required to connect with the broker and Microbit, respectively. The client variable connects to the web socket client demo set up by HiveMQ, while the global variables below that control the state of the LED and the service type, 2 being the LED, 3 being heading of the device, and 4 the acceleration.

```
var mqtt    = require('mqtt');
var noble   = require('noble');
var client  = mqtt.connect('mqtt://broker.mqttdashboard.com');
var ledState = false, serviceNo = 2
console.log("Started");
```

To initialise the events, the following four lines must be included to begin the transmission of data to and from the gateway. For Noble, the “stageChange” is called to check if the device has power and discover is used to find the peripheral. For the MQTT broker, “message” allows messages to be inputted to the script, the connect allows connection to the topic.

```
noble.on('stateChange', stateChangeEventHandler);
noble.on('discover', discoverDeviceEventHandler);
client.on('message' , messageCallBack);
client.on('connect' , connectCallBack);
```

The messageCallback function points the user in the right direction and sets the global variable appropriately to allow the subsequent functions to use them. The serviceNo is dependent on input from the user, the input is based upon the need for information from the device, if the device needs to be found, the ledState is set to allow the LED to turn on.

```
function messageCallBack(topic, message) {
    console.log("Message received");
    if(message == 'baby where?') {
        ledState = true, serviceNo = 2;
        console.log("LED is on");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    } else if(message == 'baby found')  {
        ledState = false, serviceNo = 2;
        console.log("LED is off");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    }else if(message == 'baby heading?') {
        serviceNo = 3;
        console.log("Mag");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    } else if(message == 'baby good?') {
        serviceNo = 4;
        console.log("Accel");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    } else if(message == 'exit') {
        peripheralGlobal.disconnect(disconnectCallback);
        console.log("Disconnected");
    }
}
```

The function above can call either the disconnectCallback which does exactly as the title describes, and the discoverServicesCallback, which is passed the serviceNo to allow the correct service to be selected. The discoverCharsCallback is called to configure those services.

```
function discoverServicesCallback(error, services) {
    if (error) {
        console.log("error discovering services");
    } else {
        console.log("Services");
        var deviceInformationService = services[serviceNo];
        deviceInformationService.discoverCharacteristics(null, discoverCharsCallback);
    }
}
```

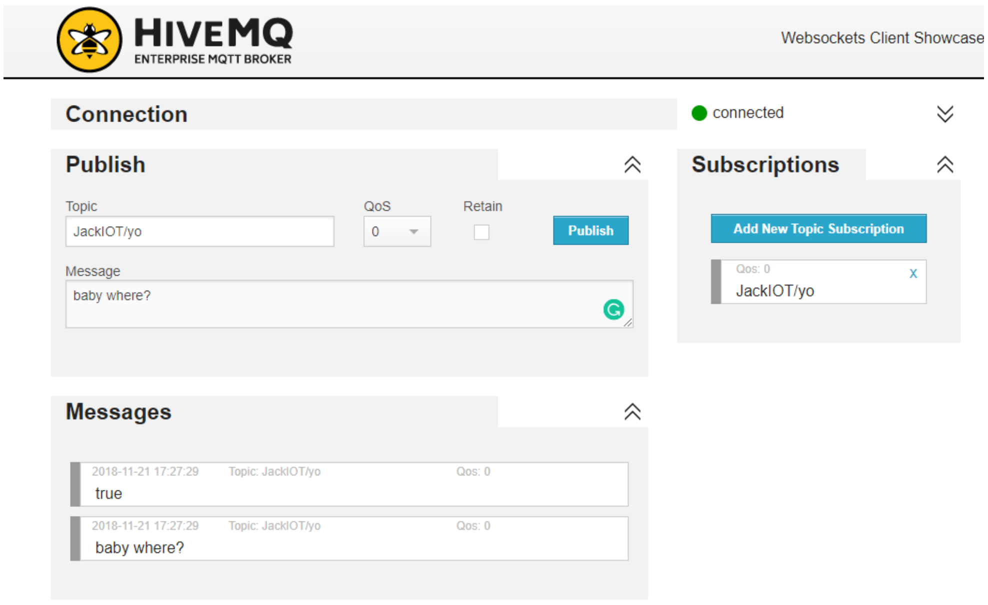
When no error occurs, the discoverCharsCallback first checks for the serviceNo, in the case of the LED, the state is checked and a buffer is made inside the write function which allows for the LED pin on the Microbit to be accessed, after typecasting the ledState to a string, the state of the pin is then published to the broker as a Boolean to a string value. If the serviceNo is either 3 or 4, a loop iterates through the three characteristics, X, Y, Z, for each characteristics the readAccel/readMag is function is called to print each value, then subscribes to the desired topic.

```
function discoverCharsCallback(error, characteristics) {
    if (error) {
        console.log("error discovering characteristics");
    } else {
        if(serviceNo == 2) {
            var sensorLevelData = characteristics[0];
            if(ledState == true) {
                sensorLevelData.write(new Buffer([ledState]), false , writeCallBackError);
            } else {
                sensorLevelData.write(new Buffer([ledState]), false , writeCallBackError);
            }
            var ledString = ledState.toString();
            client.publish('JackIOT/yo', ledString, publishCallBack);
            client.unsubscribe('JackIOT/yo');
            client.subscribe('JackIOT/yo');
        } else if(serviceNo == 3) {
            for (var i in characteristics) {
                var sensorLevelData = characteristics[i];
                sensorLevelData.read(readMag);
                client.unsubscribe('JackIOT/yo');
                client.subscribe('JackIOT/yo');
            }
        } else if(serviceNo == 4) {
            for (var i in characteristics) {
                var sensorLevelData = characteristics[i];
                sensorLevelData.read(readAccel);
                client.unsubscribe('JackIOT/yo');
                client.subscribe('JackIOT/yo');
            }
        }
    }
}
```

The readMag and readAccel are identical in parameters and function calls, with and exception for the publish data with the direction and acceleration being printed separately. The accel variable is a result of a typecasting of the data parameter in hex form, this is followed by publishCallBack function.

Current Solution

The screen captures shown below are the baby monitor in action on the MQTT broker, the user input is based upon "baby" followed by the request, this then corresponds to the messageCallBack function where the requests are linked to the global variables.



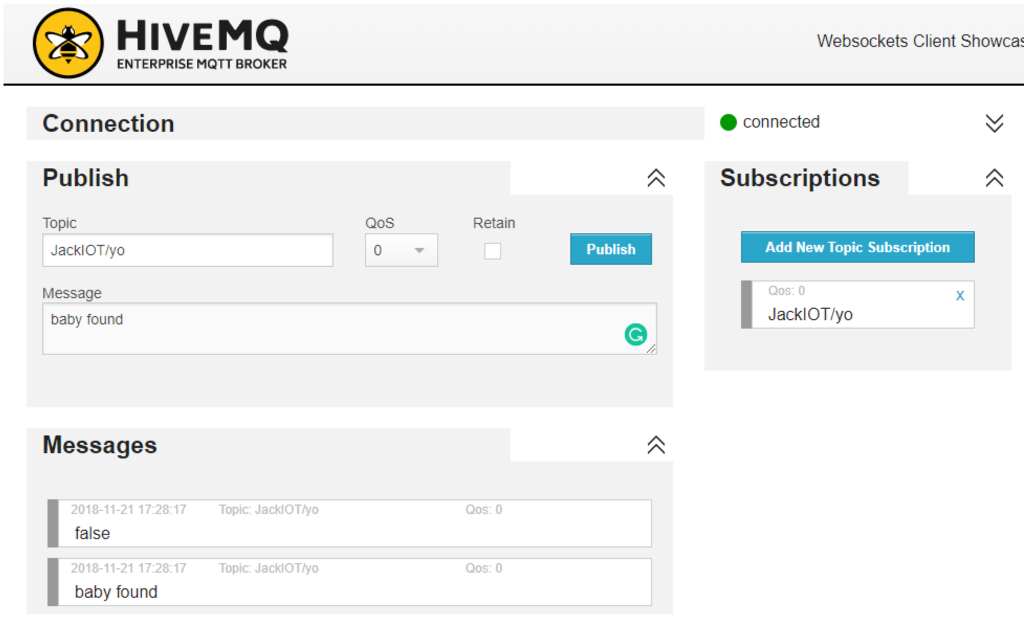
Topic: JackIOT/yo
QoS: 0
Retain:
Message: baby where?

Subscriptions: Qos: 0 JackIOT/yo

```
2018-11-21 17:27:29    Topic: JackIOT/yo    Qos: 0
true
2018-11-21 17:27:29    Topic: JackIOT/yo    Qos: 0
baby where?
```

Figure 3 LED on (Service = 2, ledState = true)

"baby where?" is used when the user wants to find the device by lighting an LED, the value of the LED is also returned to the user.



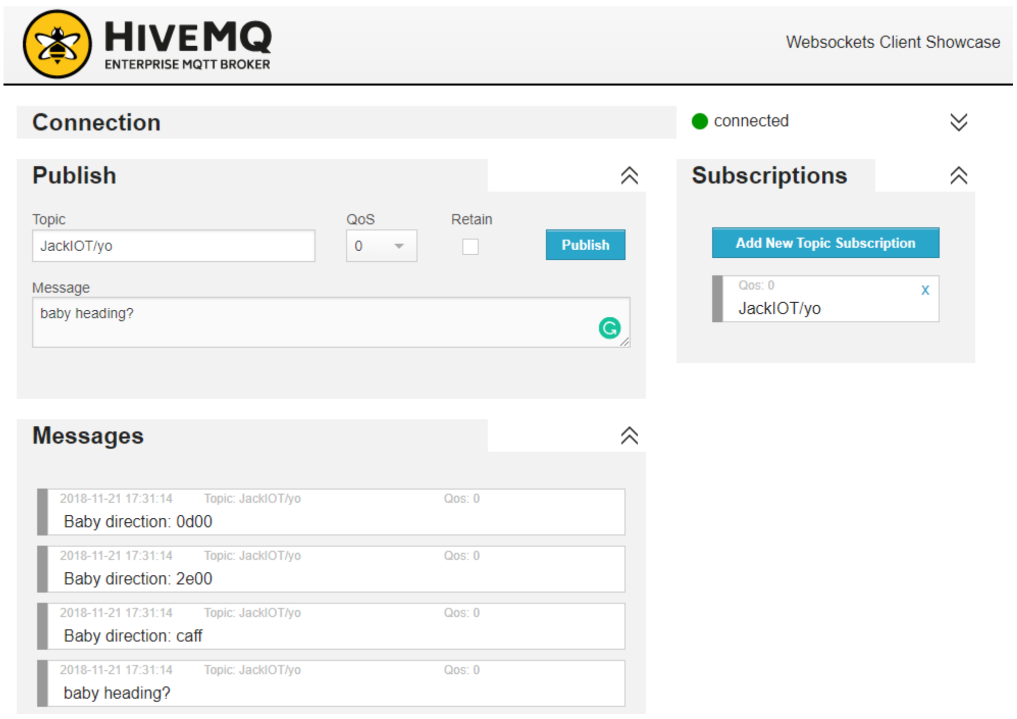
Topic: JackIOT/yo
QoS: 0
Retain:
Message: baby found

Subscriptions: Qos: 0 JackIOT/yo

```
2018-11-21 17:28:17    Topic: JackIOT/yo    Qos: 0
false
2018-11-21 17:28:17    Topic: JackIOT/yo    Qos: 0
baby found
```

Figure 4 LED on (Service = 2, ledState = false)

"baby found" is used when the user has found the device and needs to switch off the LED, the value of the LED is also returned to the user.



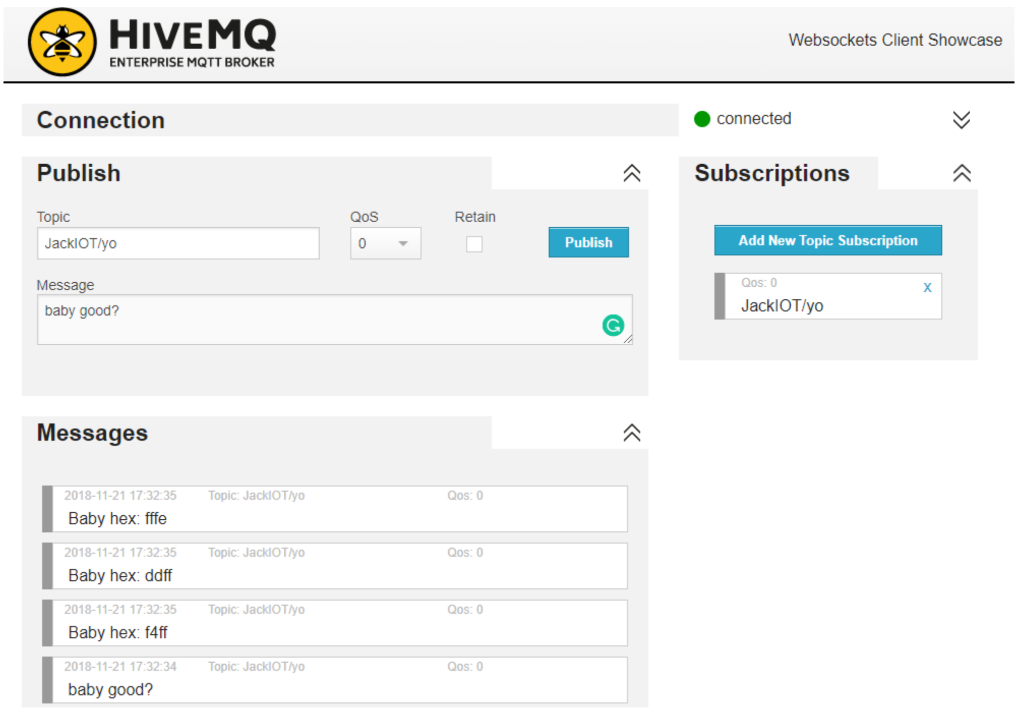
Topic: JackIOT/yo
QoS: 0
Retain: ☐
Message: baby heading?

Qos: 0
JackIOT/yo

```
2018-11-21 17:31:14    Topic: JackIOT/yo    Qos: 0
Baby direction: 0d00
2018-11-21 17:31:14    Topic: JackIOT/yo    Qos: 0
Baby direction: 2e00
2018-11-21 17:31:14    Topic: JackIOT/yo    Qos: 0
Baby direction: caff
2018-11-21 17:31:14    Topic: JackIOT/yo    Qos: 0
baby heading?
```

Figure 5 Device Heading (Service = 3)

“baby heading?” is used when the user wants to find the heading (Magnetometer value) of the device, each characteristic (X, Y, Z) is returned as a hex value.



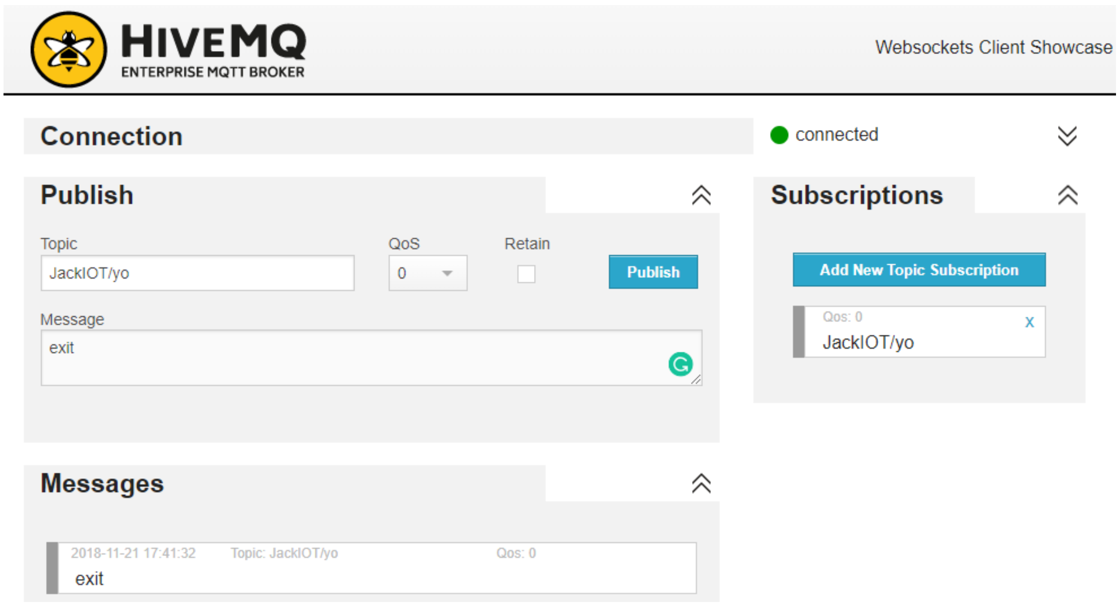
Topic: JackIOT/yo
QoS: 0
Retain: ☐
Message: baby good?

Qos: 0
JackIOT/yo

```
2018-11-21 17:32:35    Topic: JackIOT/yo    Qos: 0
Baby hex: fffe
2018-11-21 17:32:35    Topic: JackIOT/yo    Qos: 0
Baby hex: ddff
2018-11-21 17:32:35    Topic: JackIOT/yo    Qos: 0
Baby hex: f4ff
2018-11-21 17:32:34    Topic: JackIOT/yo    Qos: 0
baby good?
```

Figure 6 Device Acceleration (Service = 4)

“baby good?” is used when the user wants to determine the wellbeing of the device (Accelerometer value) of the device, each characteristic (X, Y, Z) is returned as a hex value.



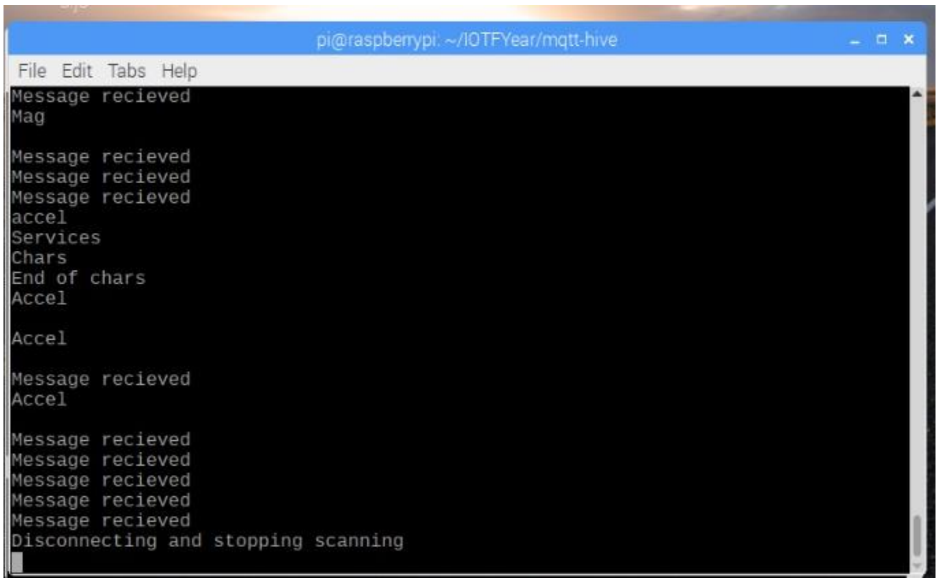
Topic: JackIOT/yo
QoS: 0
Retain
Message: exit

Qos: 0
JackIOT/yo

2018-11-21 17:41:32 Topic: JackIOT/yo Qos: 0
exit

Figure 7 User Exit

The user no longer requires the service.



```
Message recieved
Mag

Message recieved
Message recieved
Message recieved
accel
Services
Chars
End of chars
Accel

Accel

Message recieved
Accel

Message recieved
Message recieved
Message recieved
Message recieved
Message recieved
Disconnecting and stopping scanning
```

Figure 8 Server-Side Terminal

The screen capture in Figure 8 shows the result of entering the commands in Figures 3-7. With the first five being the messages upon receival and the final one the “exit”.

Proposed Solution

The current solution is not entirely user friendly with specific commands to be typed in to achieve basic results, the future implementation could add to the experience and only require buttons to be pressed. In addition to the user experience a database could be used to store the previous readings of the services, this data could be used to add some further intelligence to the system.

Full Code

```
var mqtt    = require('mqtt');                                             // includes npm libraries
var noble   = require('noble');
var client  = mqtt.connect('mqtt://broker.mqttdashboard.com');
var ledState = false, serviceNo = 2;                                       // ledState (on/off), serviceNo (LED/Mag/Accel)
console.log("Started");

/* *************** Events **************** */

noble.on('stateChange', stateChangeEventHandler);                          // calls to determine state change
noble.on('discover', discoverDeviceEventHandler);                          // find device callback
client.on('message' , messageCallBack);                                    // accepts a message, necessary for input
client.on('connect' , connectCallBack);                                    // initialises connection with MQTT

/* *************** MQTT **************** */

function messageCallBack(topic, message) {                                 // this determines what the user wants to do
    console.log("Message received");
    if(message == 'baby where?') {                                         // if this string is 'led on'
        ledState = true, serviceNo = 2;                                    // led switched on, led service selected
        console.log("LED is on");
        peripheralGlobal.discoverServices([], discoverServicesCallback);// calls function to sort services
    } else if(message == 'baby found')  {
        ledState = false, serviceNo = 2;                                   // state is off, in LED service
        console.log("LED is off");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    }else if(message == 'baby heading?') {                                 // what direction is device?
        serviceNo = 3;                                                     // magService selected
        console.log("Mag");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    } else if(message == 'baby good?') {                                   // acceleration of device
        serviceNo = 4;                                                     // accelService selected
        console.log("Accel");
        peripheralGlobal.discoverServices([], discoverServicesCallback);
    } else if(message == 'exit') {                                         // the user wants to exit
        peripheralGlobal.disconnect(disconnectCallback);
        console.log("Disconnected");
    }
}

function publishCallBack(error) {
    if (error) {
        console.log("error publishing data");
    } else {
        console.log("    ");
    }
}

/* *************** Noble **************** */

function stateChangeEventHandler(state) {                                  //event handler callback function
    if (state === 'poweredOn') {                                           // is device powered on
        console.log("starting scanning");
        noble.startScanning();                                             // start scan
    } else {                                                               // if not on
        console.log("stopping scanning");
        noble.stopScanning();                                              // stop scan
    }
}

function discoverDeviceEventHandler(peripheral) {              //event handler callback function for when discover event occurs
    console.log('Found device: ' + peripheral.advertisement.localName); // logs Mufasa
    console.log("Peripheral UUID: " + peripheral.uuid);
    if (peripheral.uuid == "f00a6eae7c20") {                               // MAC Addr of micro
        peripheralGlobal = peripheral;
        peripheral.connect(connectCallback);                             // calls connectCallback to subscribe subscribe
    }
}

function connectCallback(error) {                                          // returns device name
    if (error) {
        console.log("error connecting to peripheral");
    } else {
        console.log('Connected to peripheral: ' + "    " + peripheralGlobal.advertisement.localName);
        peripheralGlobal.discoverServices([], discoverServicesCallback); // jumps to services function
    }
}

function discoverServicesCallback(error, services) {                       // takes services and sends to characteristics
    if (error) {
        console.log("error discovering services");
    } else {
        console.log("Services");
        var deviceInformationService = services[serviceNo];          // selecting a service in the device, specified by number
        deviceInformationService.discoverCharacteristics(null, discoverCharsCallback); // calls chars callback
    }
}

function discoverCharsCallback(error, characteristics) {                   // used to decide which service to display
```

```
    if (error) {
        console.log("error discovering characteristics");
    } else {
        if(serviceNo == 2) {                                        // is it service 2
            var sensorLevelData = characteristics[0];
            if(ledState == true) {                                  // is the led set
                sensorLevelData.write(new Buffer([ledState]), false , writeCallBackError);// change ledState to on
            } else {
                sensorLevelData.write(new Buffer([ledState]), false , writeCallBackError);// change ledState to off
            }
            var ledString = ledState.toString();                    // converts bool to string
            client.publish('JackIOT/yo', ledString, publishCallBack);   // publishes state of LED
            client.unsubscribe('JackIOT/yo');
            client.subscribe('JackIOT/yo');
        } else if(serviceNo == 3 || serviceNo == 4) {               // if direction is selected (service 3)
            for (var i in characteristics) {                        // iterates through characteristics to print X, Y, Z
                var sensorLevelData = characteristics[i];
                if(serviceNo == 4){
                    sensorLevelData.read(readAccel);
                } else{
                    sensorLevelData.read(readMag);                 // jumps to readMag
                }
                client.unsubscribe('JackIOT/yo');
                client.subscribe('JackIOT/yo');
            }
        }
    }
}

function readMag(error, data) {                                     // prints mag data
    if (error) {
        console.log("error reading data");
    } else {
        var mag = data.toString('hex');                             // converts X, Y, Z on each iteration of for loop
        client.publish('JackIOT/yo', 'Baby direction: ' + mag, publishCallBack); // publishes the heading of device
    }
}

function readAccel(error, data) {                                   // prints accel data
    if (error) {
        console.log("error reading data");
    } else {
        var accel = data.toString('hex');                           // same as readMag
        client.publish('JackIOT/yo', 'Baby hex: ' + accel, publishCallBack); // publishes acceleration of device
    }
}

function writeCallBackError(error) {                                // logs error when called
    if(error) {
        console.log("write error");
    }
}

function disconnectCallback(error) {                           // this will be executed when the disconnect request returns
    if (error) {
        console.log("error disconnecting");
    } else {
        console.log("Disconnecting and stopping scanning");
    }
}
```