

Watson IoT Platform – T3 - NodeRed – Device – connection

Commissioning task 3

1. Introduction

In this commissioning task we will create a connection between a real device and the IBM Watson NodeRED cloud application.

As well we will create a user interface which works on a web browser.

In the commissioning task T0 you created for yourself a user id in the IBM Cloud environment, you created a device and wrote device data either with a MQTT client application or with a micro controller.

If you did not already do the above mentioned tasks please open the relevant instructions in the IBM Cloud documentation. While writing the latest version of this instruction it was possible to open the latest link into Watson IoT Platform getting started guides in the IBM Cloud web page

<https://www.ibm.com/cloud/get-started/iot-platform>

If you have got about one hour time and you are ready to register as a user please select "Connect a networked device to the IBM Watson IoT Platform". Further please follow the link "Play with IBM Watson IoT Platform".

In the commissioning task T2 we processed device data in the Watson IoT Platform. For the processing we created an application with the Node-RED editor. The application was saved on the CloudantNoSQLDB data base.

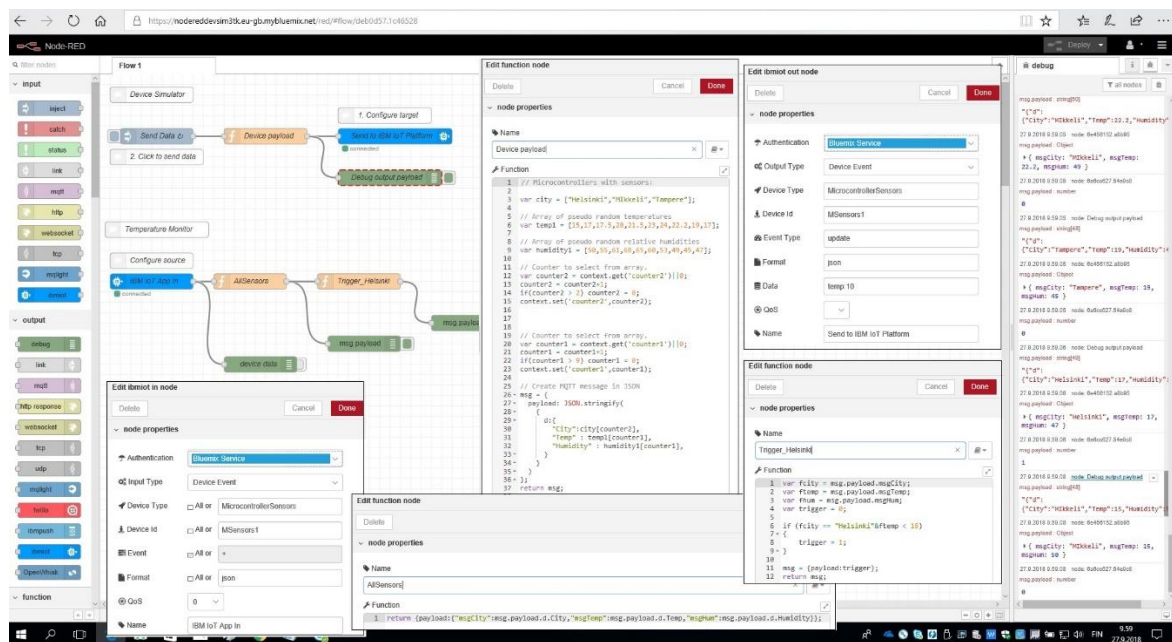


Fig. 1.1 NodeRED –flow in the commissioning task T2 and the content for each node

If you haven't created the NodeRED flow already it would not take too long to type the required content into the node simulating the device data and into the node receiving the device data.

Exactly same device type and exactly same device id needs to be created into exactly the same organization id.

The screenshot shows the 'Browse Devices' interface in the IBM Watson IoT Platform. The page has a dark header with the IBM Watson IoT Platform logo and a user profile. Below the header, there's a navigation bar with 'Browse', 'Action', and 'Device Types'. A search bar is present on the right. The main content area is titled 'Browse Devices' and includes a sub-header 'All Devices'. A paragraph explains that the table shows a summary of added devices and can be filtered, organized, and searched. Below this is a table with 3 results. The table has columns for Device ID, Device Type, Class ID, and Date Added. The results are MSensors1, MServos1, and MServos2, all of type MicrocontrollerSensors or MicrocontrollerServos.

Device ID	Device Type	Class ID	Date Added
MSensors1	MicrocontrollerSensors	Device	27 Sep 2018 14:36
MServos1	MicrocontrollerServos	Device	27 Sep 2018 14:43
MServos2	MicrocontrollerServos	Device	28 Sep 2018 09:57

Fig 1.2 The devices from the Commissioning Task 2 seen on the Device View.

Messages to the device – real or simulated – can be created with the flow seen below.

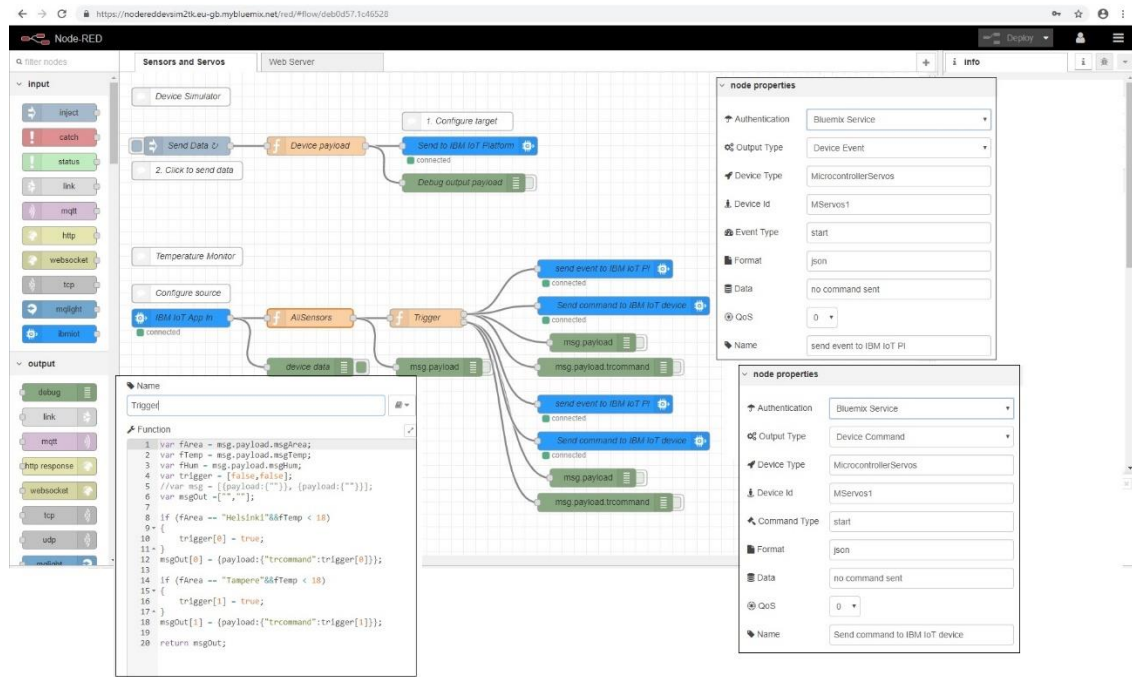


Fig. 1.3 Creating message to be sent to devices from the Commissioning Task 2.

The flow seen on picture above can be imported to your own Watson Node-RED editor with the Import Clipboard function. Please copy the following “code” into the clipboard. Paste it to the Windows Notepad editor for removing the invisible control symbols. Copy it again from the Notepad and paste it to the Node-RED.

```
{
  "id": "deb0d57.1c46528",
  "type": "tab",
  "label": "Sensors and Servos",
  "disabled": false,
  "info": "",
  "id": "3e77d543.c1882a",
  "type": "ibmiot
in",
  "z": "deb0d57.1c46528",
  "authentication": "boundService",
  "apiKey": "",
  "inputType": "evt",
  "logicalInterface": "",
  "ruleId": "",
  "deviceId": "MSensors1",
  "ap
plicationId": "",
  "deviceType": "MicrocontrollerSensors",
  "eventType": "+",
  "commandType": "",
  "format": "json",
  "name": "IBM IoT App
In",
  "service": "registered",
  "allDevices": false,
  "allApplications": false,
  "allDeviceTypes": false,
  "allLogicalInterfaces": false,
  "allEvents": true,
  "allCommands": false,
  "allFormats": false,
  "qos": "0",
  "x": "100",
  "y": "400",
  "wires": [
    [
      "ae0082ac.51ff8",
      "c0c482df.3f3b8"
    ]
  ],
  "id": "ae0082ac.51ff8",
  "type": "function",
  "z": "deb0d57.1c
46528",
  "name": "AllSensors",
  "func": "return
{payload:{\n\"msgArea\":msg.payload.d.Area,\n\"msgTemp\":msg.payload.d.Temp,\n\"msgHum\":msg.payload.d.Humidity}};\n\",
  \"outputs\":1,\n\"noerr\":0,\n\"x\":310,\n\"y\":400,\n\"wires\":[[\"54b28de8.4a9c34\",
  \"82c0704.645f29\"]];\n\",
  \"id\": \"c0c482df.3f3b8\",
  \"type\": \"debug\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"device
data\",
  \"active\": true,
  \"console\": false,
  \"complete\": true,
  \"x\": \"310\",
  \"y\": \"480\",
  \"wires\": [
    [
      \"cbe1a0b7.cd877\"
    ]
  ],
  \"id\": \"cbe1a0b7.cd877\",
  \"type\": \"ibmiot
out\",
  \"z\": \"deb0d57.1c46528\",
  \"authentication\": \"boundService\",
  \"apiKey\": \"\",
  \"outputType\": \"evt\",
  \"deviceId\": \"MSensors1\",
  \"deviceType\": \"MicrocontrollerSe
nsors\",
  \"eventCommandType\": \"update\",
  \"format\": \"json\",
  \"data\": \"temp:10\",
  \"qos\": \"\",
  \"name\": \"Send to IBM IoT
Platform\",
  \"service\": \"registered\",
  \"x\": \"570\",
  \"y\": \"120\",
  \"wires\": [
    [
      \"5917a925.6a3c08\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"5917a925.6a3c08\",
  \"type\": \"inject\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Send
Data\",
  \"topic\": \"\",
  \"payload\": true,
  \"payloadType\": \"bool\",
  \"repeat\": \"60\",
  \"crontab\": \"\",
  \"once\": false,
  \"onceDelay\": \"\",
  \"x\": \"110\",
  \"y\": \"120\",
  \"wires\": [
    [
      \"6b4a591c.014b18\"
    ]
  ],
  \"id\": \"6b4a591c.014b18\",
  \"type\": \"function\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Device payload\",
  \"func\": \"// Microcontrollers with sensors\n\n\nvar
area1 = [\n\"Greenhouse1\",
\n\"Greenhouse2\",
\n\"Greenhouse3\"];\n\n// Array of pseudo random temperatures\n\nvar temp1 =
[15,17,17.5,20,21,23,24,22.2,19,17];\n\n// Array of pseudo random relative humidities\n\nvar humidity1 = [50,55,61,68,65,60,53,49,45,47];\n\n//
Counter to select from array.\n\nvar counter1 = context.get('counter1') || 0;\n\ncounter1 = counter1+1;\n\nif(counter1 > 9) counter1 =
0;\n\ncontext.set('counter1',counter1);\n\n// Counter to select from array.\n\nvar counter2 = context.get('counter2') || 0;\n\ncounter2 =
counter2+1;\n\nif(counter2 > 2) counter2 = 0;\n\ncontext.set('counter2',counter2);\n\n// Create MQTT message in JSON\n\nmsg = {\n\n  payload:
JSON.stringify(\n\n    {\n\n      d: {\n\n        \"Area\": area1[counter2],\n\n        \"Temp\" : temp1[counter1],\n\n        \"Humidity\": humidity1[counter1],\n\n      }\n\n    }\n\n  )\n\nreturn
msg;\n\n\",
  \"outputs\":1,\n\"noerr\":0,\n\"x\":320,\n\"y\":120,\n\"wires\":[[\"cbe1a0b7.cd877\",
  \"805c97ee.3ed9e8\"]],\n\"id\": \"805c97ee.3ed9e8\",
  \"type\": \"debug\",
  \"z\": \"deb0d5
7.1c46528\",
  \"name\": \"Debug output
payload\",
  \"active\": false,
  \"tosidebar\": true,
  \"console\": false,
  \"complete\": \"payload\",
  \"x\": \"560\",
  \"y\": \"180\",
  \"wires\": [
    [
      \"86df0b6c.af90c8\"
    ]
  ],
  \"id\": \"86df0b6c.af90c8\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Device Simulator\",
  \"info\": \"Sends simulated device sensor data to IBM Watson IoT Platform.\n\n\nCan be configured to send
on click or on an automatic interval.\n\n\n#Prerequisite\n\nOutput node device type and device ID need to match a device that it registered in a running
IBM Watson IoT Platform service.\n\n\n# Watson IoT Platform docs\n\n[Connecting
devices](https://www.ibm.com/docs/services/iot/iotplatform_task.html)\",
  \"x\": \"100\",
  \"y\": \"40\",
  \"wires\": [
    [
      \"141b7c7d.ad2a84\"
    ]
  ],
  \"id\": \"141b7c7d.ad2a84\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"1. Configure
target\",
  \"info\": \"\",
  \"x\": \"550\",
  \"y\": \"80\",
  \"wires\": [
    [
      \"c2dd8ed5.7dd7f\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"c2dd8ed5.7dd7f\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"2. Click to send data\",
  \"info\": \"To
automatically send data:\n1. Change *Repeat* to interval.\n2. Click Deploy
button.\n\n\",
  \"x\": \"110\",
  \"y\": \"160\",
  \"wires\": [
    [
      \"7926c7b2.86d938\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"7926c7b2.86d938\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Temperature
Monitor\",
  \"info\": \"\",
  \"x\": \"110\",
  \"y\": \"300\",
  \"wires\": [
    [
      \"188a5e87.e775a1\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"188a5e87.e775a1\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Configure
source\",
  \"info\": \"\",
  \"x\": \"100\",
  \"y\": \"360\",
  \"wires\": [
    [
      \"54b28de8.4a9c34\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"54b28de8.4a9c34\",
  \"type\": \"function\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Trigger\",
  \"func\": \"var fArea =
msg.payload.msArea;\n\nvar fTemp = msg.payload.msTemp;\n\nvar fHum = msg.payload.msHum;\n\nvar trigger = false;\n\nif(fArea != false && fTemp != false && fHum != false) {\n\n  trigger = true;\n\n}\n\nreturn
trigger;\n\n\",
  \"outputs\":1,\n\"noerr\":0,\n\"x\":330,\n\"y\":180,\n\"wires\":[[\"cbe1a0b7.cd877\",
  \"805c97ee.3ed9e8\"]],\n\"id\": \"cbe1a0b7.cd877\",
  \"type\": \"ibmiot
out\",
  \"z\": \"deb0d57.1c46528\",
  \"authentication\": \"boundService\",
  \"apiKey\": \"\",
  \"outputType\": \"evt\",
  \"deviceId\": \"MSensors1\",
  \"deviceType\": \"MicrocontrollerSensors\",
  \"eventCommandType\": \"update\",
  \"format\": \"json\",
  \"data\": \"temp:10\",
  \"qos\": \"\",
  \"name\": \"Send to IBM IoT
Platform\",
  \"service\": \"registered\",
  \"x\": \"570\",
  \"y\": \"120\",
  \"wires\": [
    [
      \"5917a925.6a3c08\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"5917a925.6a3c08\",
  \"type\": \"inject\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Send
Data\",
  \"topic\": \"\",
  \"payload\": true,
  \"payloadType\": \"bool\",
  \"repeat\": \"60\",
  \"crontab\": \"\",
  \"once\": false,
  \"onceDelay\": \"\",
  \"x\": \"110\",
  \"y\": \"120\",
  \"wires\": [
    [
      \"6b4a591c.014b18\"
    ]
  ],
  \"id\": \"6b4a591c.014b18\",
  \"type\": \"function\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Device payload\",
  \"func\": \"// Microcontrollers with sensors\n\n\nvar
area1 = [\n\"Greenhouse1\",
\n\"Greenhouse2\",
\n\"Greenhouse3\"];\n\n// Array of pseudo random temperatures\n\nvar temp1 =
[15,17,17.5,20,21,23,24,22.2,19,17];\n\n// Array of pseudo random relative humidities\n\nvar humidity1 = [50,55,61,68,65,60,53,49,45,47];\n\n//
Counter to select from array.\n\nvar counter1 = context.get('counter1') || 0;\n\ncounter1 = counter1+1;\n\nif(counter1 > 9) counter1 =
0;\n\ncontext.set('counter1',counter1);\n\n// Counter to select from array.\n\nvar counter2 = context.get('counter2') || 0;\n\ncounter2 =
counter2+1;\n\nif(counter2 > 2) counter2 = 0;\n\ncontext.set('counter2',counter2);\n\n// Create MQTT message in JSON\n\nmsg = {\n\n  payload:
JSON.stringify(\n\n    {\n\n      d: {\n\n        \"Area\": area1[counter2],\n\n        \"Temp\" : temp1[counter1],\n\n        \"Humidity\": humidity1[counter1],\n\n      }\n\n    }\n\n  )\n\nreturn
msg;\n\n\",
  \"outputs\":1,\n\"noerr\":0,\n\"x\":320,\n\"y\":120,\n\"wires\":[[\"cbe1a0b7.cd877\",
  \"805c97ee.3ed9e8\"]],\n\"id\": \"805c97ee.3ed9e8\",
  \"type\": \"debug\",
  \"z\": \"deb0d5
7.1c46528\",
  \"name\": \"Debug output
payload\",
  \"active\": false,
  \"tosidebar\": true,
  \"console\": false,
  \"complete\": \"payload\",
  \"x\": \"560\",
  \"y\": \"180\",
  \"wires\": [
    [
      \"86df0b6c.af90c8\"
    ]
  ],
  \"id\": \"86df0b6c.af90c8\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"Device Simulator\",
  \"info\": \"Sends simulated device sensor data to IBM Watson IoT Platform.\n\n\nCan be configured to send
on click or on an automatic interval.\n\n\n#Prerequisite\n\nOutput node device type and device ID need to match a device that it registered in a running
IBM Watson IoT Platform service.\n\n\n# Watson IoT Platform docs\n\n[Connecting
devices](https://www.ibm.com/docs/services/iot/iotplatform_task.html)\",
  \"x\": \"100\",
  \"y\": \"40\",
  \"wires\": [
    [
      \"141b7c7d.ad2a84\"
    ]
  ],
  \"id\": \"141b7c7d.ad2a84\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",
  \"name\": \"1. Configure
target\",
  \"info\": \"\",
  \"x\": \"550\",
  \"y\": \"80\",
  \"wires\": [
    [
      \"c2dd8ed5.7dd7f\",
      \"c0c482df.3f3b8\"
    ]
  ],
  \"id\": \"c2dd8ed5.7dd7f\",
  \"type\": \"comment\",
  \"z\": \"deb0d57.1c46528\",

```

```
{[payload:{\"\\\"}, {payload:{\"\\\"}};\\nvar msgOut=[\"\\\", \"\\\"]:\\n\\nif (fArea == '\\\"Helsinki\\\"' && fTemp < 18)\\n{\\n    trigger[0] = true;\\n}\\nmsgOut[0] =  
[payload:{\"\\\"trcommand\\\":trigger[0]}];\\n\\nif (fArea == '\\\"Tampere\\\"' && fTemp < 18)\\n{\\n    trigger[1] = true;\\n}\\nmsgOut[1] =  
[payload:{\"\\\"trcommand\\\":trigger[1]}];\\n}\\nreturn  
msgOut;}\", \"outputs\":\"2,\"noerr\":\"0\",\"x\":\"500,\"y\":\"400,\"wires\":[{\"\"86eabffb.b92be\"\",\"8e9ae4a8.9edf88\"\",\"476d79d6.e80738\"\",\"a84195d8.b33de8\"}],[{\"9e624f2a.1  
e153\"\",\"21af5407.97aeac\"\",\"c4cd1.71c2af3\"\",\"351079b3.3df66\"}]]},{\"id\":\"82c0704.645f29\",\"type\":\"debug\",\"z\":\"deb0d57.1c46528\",\"name\":\"\", \"active\":  
false,\"tosidebar\":true,\"console\":false,\"tostatus\":false,\"complete\":false,\"x\":\"520,\"y\":\"480,\"wires\":[{}],\"id\":\"86eabffb.b92be\",\"type\":\"debug\",\"z\":\"deb0d5  
7.1c46528\",\"name\":\"\", \"active\":false,\"tosidebar\":true,\"console\":false,\"tostatus\":false,\"complete\":false,\"x\":\"770,\"y\":\"440,\"wires\":[{}],\"id\":\"8e9ae4a8.9ed  
f88\",\"type\":\"ibmiot  
out\",\"z\":\"deb0d57.1c46528\",\"authentication\":\"boundService\",\"apiKey\":\"\", \"outputType\":\"evt\",\"deviceId\":\"MServos1\",\"deviceType\":\"MicrocontrollerSer  
vos\",\"eventCommandType\":\"start\",\"format\":\"json\",\"data\":\"no command sent\", \"qos\":\"0,\"name\":\"Send event to IBM IoT  
PI\",\"service\":\"registered\", \"x\":\"790,\"y\":\"320,\"wires\":[{}],\"id\":\"476d79d6.e80738\",\"type\":\"debug\",\"z\":\"deb0d57.1c46528\",\"name\":\"\", \"active\":false,\"tosideb  
ar\":false,\"console\":false,\"tostatus\":false,\"complete\":false,\"payload.trcommand\",\"x\":\"790,\"y\":\"480,\"wires\":[{}],\"id\":\"9e624f2a.1e153\",\"type\":\"ibmiot  
out\",\"z\":\"deb0d57.1c46528\",\"authentication\":\"boundService\",\"apiKey\":\"\", \"outputType\":\"evt\",\"deviceId\":\"MServos2\",\"deviceType\":\"MicrocontrollerSer  
vos\",\"eventCommandType\":\"start\",\"format\":\"json\",\"data\":\"no command sent\", \"qos\":\"0,\"name\":\"Send event to IBM IoT  
PI\",\"service\":\"registered\", \"x\":\"790,\"y\":\"540,\"wires\":[{}],\"id\":\"21af5407.97aeac\",\"type\":\"debug\",\"z\":\"deb0d57.1c46528\",\"name\":\"\", \"active\":false,\"tosideb  
ar\":true,\"console\":false,\"tostatus\":false,\"complete\":false,\"x\":\"770,\"y\":\"660,\"wires\":[{}],\"id\":\"c4cd1.71c2af3\",\"type\":\"debug\",\"z\":\"deb0d57.1c46528\"],  
\"name\":\"\", \"active\":false,\"tosidebar\":true,\"console\":false,\"tostatus\":false,\"complete\":\"payload.trcommand\", \"x\":\"790,\"y\":\"700,\"wires\":[{}],\"id\":\"a84195d8.  
b33de8\", \"type\":\"ibmiot  
out\",\"z\":\"deb0d57.1c46528\",\"authentication\":\"boundService\",\"apiKey\":\"\", \"outputType\":\"cmd\",\"deviceId\":\"MServos1\", \"deviceType\":\"MicrocontrollerSe  
rvos\",\"eventCommandType\":\"start\",\"format\":\"json\", \"data\":\"no command sent\", \"qos\":\"0,\"name\":\"Send command to IBM IoT  
device\", \"service\":\"registered\", \"x\":\"820,\"y\":\"380,\"wires\":[{}],\"id\":\"351079b3.3df66\", \"type\":\"ibmiot  
out\", \"z\":\"deb0d57.1c46528\", \"authentication\":\"boundService\", \"apiKey\":\"\", \"outputType\":\"cmd\", \"deviceId\":\"MServos2\", \"deviceType\":\"MicrocontrollerSe  
rvos\", \"eventCommandType\":\"start\", \"format\":\"json\", \"data\":\"no command sent\", \"qos\":\"0,\"name\":\"Send command to IBM IoT  
device\", \"service\":\"registered\", \"x\":\"820,\"y\":\"600,\"wires\":[{}]
```

The code is as well available in file

IBMWatsonSignalsFunctionsServo12_flow_va.txt

You might not be able to select the authentication method “Bluemix Service” in the Node-RED IBM IoT node. This depends on the user rights on your IBM ID. You might try with the authentication method “API key”. Browse again into the page of your Watson IoT Platform and create the API Key there. Fill in the corresponding information into the IBM IoT node.

2. Connection between your device and the Watson IoT Platform

Please create into the device view on picture 1.2. – or actually into your own device view in the Watson IoT Application Instance – a real device. First you need to create for the device:

Device Type – for example M_Sensors

Device ID – for example M_Sensor_01

and let the system create the Authentication Token. Please save into a text file the organization ID Authentication Token.... all the registering information to be used later.

The real device can be almost any device capable of sending to the Internet with http communication the messages according to the MQTT protocol.

There are detailed instruction on how to create a device and how to get the Watson IoT platform communication working in the document

“IoT_IBMWatson_T0_MKR1000_Aloitus_GettingStarted_FI.pdf” .

MQTT Box application is an application used as a mqtt client in a Windows computer. The use of MQTT Box as a “device” is explained in the end of document “”

IoT_IBMWatson_T1_mqtt_client_FI.pdf” . You can create for example the following registration data:

Device Type – for example MQTTClient_Sensors

Device ID – for example MQTTclient_Sensor_01

In this example the Arduino MKR1000 and a previously interfaced sensor is used as a device.

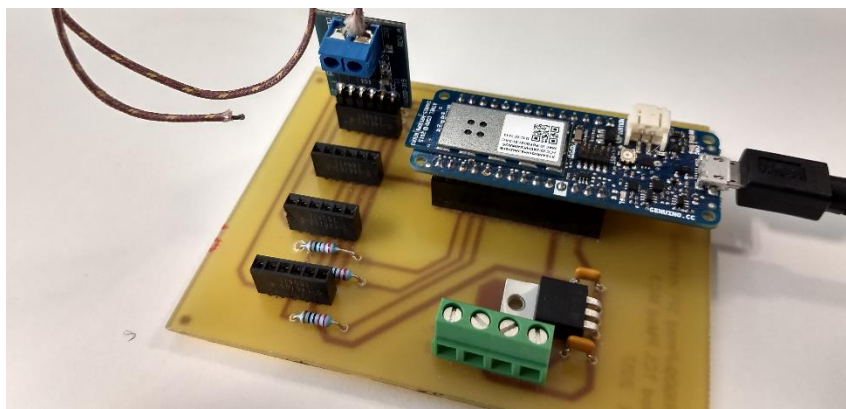


Fig. 2.1. Arduino MKR1000 and Digilent pmodTC1 –sensor module.

The following code can be used as a prototype for the software in Arduino. The code transmits sensor temperature value to the IoT Platform.

```
/*
MKR1000 connecting to IBM Watson IoT Platform

Based on documentation and "recipes" on IBM Bluemix
https://www.ibm.com/cloud-computing/bluemix/watson
Timo Karppinen 19.2.2017

Modified for testing SPI thermocouple board Digilent PmodTC1
Please connect
MKR1000 - PmodTC1
GND - 5 GND
Vcc - 6 Vcc
9 SCK - 4 SCK
10 MISO - 3 MISO
2 - 1 SS

Thermocouple data on the SPI
D31 - sign
D30 ....D18 - 13 bits of temperature data
D16 - normally FALSE. TRUE if thermocouple input is open or shorted to GND or VCC
D15 ... D0 - reference junction tempeprature

The reference junction compensation is calculated in the IC. no need to calculate here.

Timo Karppinen 25.1.2018
```

```

*/

#include <SPI.h>
#include <WiFi101.h>
#include <WiFiSSLClient.h>
#include <MQTTClient.h>

// WLAN
//char ssid[] = "Moto_Z2_TK"; // your network SSID (name)
//char pass[] = "xxxxxxxxxx"; // your network password (use for WPA)

char ssid[] = "HAMKVisitor"; // your network SSID (name)
char pass[] = "xxxxxxxxxx"; // your network password (use for WPA)

//char ssid[] = "Nelli";
//char pass[] = "xxxxxxxxxx";

// IBM Watson
// Your organization and device needs to be registered in IBM Watson IoT Platform.
// Instruction for registering on page
// https://internetofthings.ibmcloud.com/#

//char *client_id = "d:<your Organization ID><your Device Type><your Device ID>";
char *client_id = "d:yyyyyyy:A_MKR1000:DF48";
char *user_id = "use-token-auth"; // telling that authentication will be done with token
char *authToken = "xxxxxxxxxxxxxx"; // Your IBM Watson Authentication Token

//char *ibm_hostname = "your-org-id.messaging.internetofthings.ibmcloud.com";
char *ibm_hostname = "yyyyyy.messaging.internetofthings.ibmcloud.com";

// sensors and LEDS
const int LEDPin = LED_BUILTIN; // must be a pin that supports PWM. 0...8 on MKR1000
// PModTC1
const int thermoCS = 2; // chip select for MIC3 SPI communication
int thermoByte0 = 0; // 8 bit data from TC1 board
int thermoByte1 = 0; // 8 bit data from TC1 board
int thermoByte2 = 0; // 8 bit data from TC1 board
int thermoByte3 = 0; // 8 bit data from TC1 board
int temp14bit = 0; // 14 most significant bits on a 32 bit integer
int tempRaw = 0;
float tempScaledF = 0;

int blinkState = 0;

/*use this class if you connect using SSL
 * WiFiSSLClient net;
 */
WiFiClient net;
MQTTClient MQTTc;

unsigned long lastSampleMillis = 0;
unsigned long previousWiFiBeginMillis = 0;
unsigned long lastWatsonMillis = 0;
unsigned long lastPrintMillis = 0;

void setup()
{
  pinMode(thermoCS, OUTPUT);
  digitalWrite(thermoCS, HIGH); // for not communicating with MIC3 at the moment
  Serial.begin(9600);
  delay(2000); // Wait for wifi unit to power up
  WiFi.begin(ssid, pass);
  delay(5000); // Wait for WiFi to connect
  Serial.println("Connected to WLAN");
  printWiFiStatus();

  /*
   client.begin("<Address Watson IOT>", 1883, net);
   Address Watson IOT: <WatsonIOTOrganizationID>.messaging.internetofthings.ibmcloud.com
   Example:
   client.begin("iqwckl.messaging.internetofthings.ibmcloud.com", 1883, net);
  */
  MQTTc.begin(ibm_hostname, 1883, net); // Cut for testing without Watson

  connect();

  SPI.begin();
  // Set up the I/O pins

```

```
pinMode(thermoCS, OUTPUT);
pinMode(LEDPin, OUTPUT);

}

void loop() {
  MQTTC.loop(); // Cut for testing without Watson

  // opening and closing SPI communication for reading TC1
  if(millis() - lastSampleMillis > 500)
  {
    lastSampleMillis = millis();
    SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));
    digitalWrite(thermoCS, LOW);

    thermoByte0 = SPI.transfer(0x00);
    thermoByte1 = SPI.transfer(0x00);
    thermoByte2 = SPI.transfer(0x00);
    thermoByte3 = SPI.transfer(0x00);

    digitalWrite(thermoCS, HIGH);
    SPI.endTransaction();

    thermoByte0 = thermoByte0 << 24;
    thermoByte1 = thermoByte1 << 16;

    templ4bit =( thermoByte0 | thermoByte1 );

    tempRaw = templ4bit/262144; // shifting 18 bits to right gives multiply of 0,25 degree C.
    tempScaledF = float(templ4bit/262144)/4;

  }

  // Print on serial monitor once in 1000 millisecond
  if(millis() - lastPrintMillis > 1000)
  {
    Serial.print("templ4bit  ");
    Serial.println(templ4bit, BIN);
    Serial.print("  tempScaled  ");
    Serial.println(tempRaw, BIN);
    Serial.print("  tempScaledF  ");
    Serial.println(tempScaledF);

    lastPrintMillis = millis();
  }

  // publish a message every 30 second.
  if(millis() - lastWatsonMillis > 30000)
  {
    Serial.println("Publishing to Watson...");
    if(!MQTTC.connected()) { // Cut for testing without Watson
      connect(); // Cut for testing without Watson
    } // Cut for testing without Watson
    lastWatsonMillis = millis();
    //Cut for testing without Watson

    String wpayload = "{\"d\":{\"TemperatureSensor\":\"TC1 \",\"TempScaledF3AC\":\"" +
String(tempScaledF)+ ", \"TempStreightDF48\":\"" + String(templ4bit)+"}\"}";

    MQTTC.publish("iot-2/evt/TemperatureTC1/fmt/json", wpayload);

  }

  delay(1);
}

// end of loop
}

void connect()
{
  Serial.print("checking WLAN...");
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print("."); // printing a dot every half second
    if ( millis() - previousWiFiBeginMillis > 5000) // reconnecting
    {
      previousWiFiBeginMillis = millis();
    }
  }
}
```

```
    WiFi.begin(ssid, pass);
    delay(5000); // Wait for WiFi to connect
    Serial.println("Connected to WLAN");
    printWiFiStatus();
  }
  delay(500);

}
/*
Example:
MQTTc.connect("d:iqwckl:arduino:oxigenarbpn","use-token-auth","90wT2?a*1WAMVJStb1")

Documentation:
https://console.ng.bluemix.net/docs/services/IoT/iotplatform_task.html#iotplatform_task
*/

Serial.print("\nconnecting Watson with MQTT...");
// Cut for testing without Watson
while (!MQTTc.connect(client_id,user_id,authToken))
{
  Serial.print(".");
  delay(3000);
}
Serial.println("\nconnected!");
}

// messageReceived subroutine needs to be here. MQTT client is calling it.
void messageReceived(String topic, String payload, char * bytes, unsigned int length) {
  Serial.print("incoming: ");
  Serial.print(topic);
  Serial.print(" - ");
  Serial.print(payload);
  Serial.println();
}

void printWiFiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Code 1. The Arduino MKR1000 reads from the pmodTC1 module the temperature and sends the value to the Watson IoT Platform.

You can copy the code from here. But it would be better to copy and paste the code to Windows Notepad and then again copy and paste the code to the Arduino IDE. This procedure removes the control symbols.

The code is in the file

Wat_MKR_SPI_ThermoC_H_DF48.txt

3. Commissioning tasks

Assignment 1

Please change the code so that it transmits into your own Node-RED exercise the payload with JSON formatted data

```
{d:{  
  "Area":Greenhouse1,  
  "Temp":21,  
  "Humidity":70,  
}  
}
```

In the given Node-RED example you can change the variable “city” into variable “area”. This variable will get from your device the value “Greenhouse1”.

In the Node-RED example the variable “temp” will get a real up to date sensor value from the device.

Please Note! Remember to change in the Watson IoT platform:

Dashboard

Security

Connection Security: TLS Optional

After changing you need to power off the device. After a while power it on again.

There is a property in the IBM Cloud that the same device is not allowed to try to register continuously with false security, id or token. In the Arduino code the device is set to try registering every 30 seconds if it has lost the connection. After a few minutes of trying with false information the IBM Cloud will block the device out. You need to wait some time before it will be possible to try to register again.

if you are not using a real device you can send the same information with an easy to use MQTT Client application. Mqtt client applications are available for Android, Windows, iOS, Linux, ... One example is MQTT Box for Windows.

<http://workswithweb.com/mqttbox.html#>

Please look for instruction in the document “IoT_IBMWatson_T1_mqtt_client_FI.pdf”.

In the report: A couple of lines for introduction. A couple of screen captures where the sensor data is visible in your Watson IoT platform as an event or status of your device.

Assignment 2.

Please create an other device in the Watson IoT Platform. It can be named:

Device Type – for example MQTTClient_Servos

Device ID – for example MQTTclient_Servo_01

Please note! A device registering to the IBM Watson IoT is able to subscribe in mqtt protocol only for command type messages.

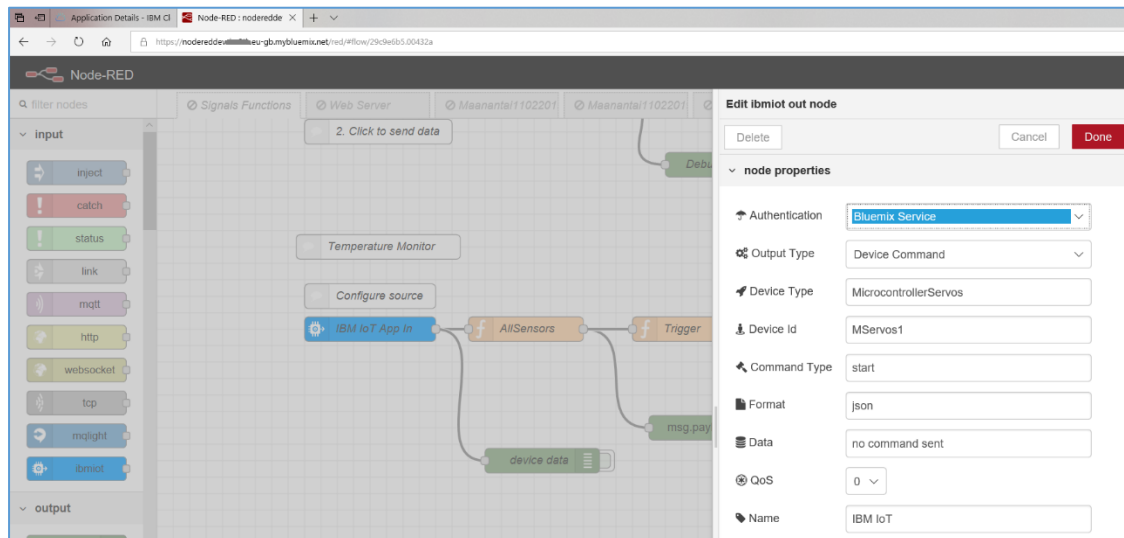


Fig. 3.1

Test the reception of messages. Please send some other messages than the ones already given in the example.

In the report: A couple of screen captures where there is an analysis result visible in Node-RED and the result initiates sending a command message. The command message should be visible in Watson IoT Device view as well.

4. A user interface created as a web page

A web page readable with a normal web browser will be created with the Node-RED editor.

There is quite a lot of code written in the flow which will create the web page. The amount of errors is minimized if you will copy the code for a starting point.

NodeRED web page flow:

```
[{"id":"ea9fc0ca.2f179","type":"websocket",
  "out","z":"91f97da0.d49da","name":"","server":"9ceb3a2f.eb53e8","client":"","x":700,"y":280,"wires":[]},
{"id":"cf29a616.572df8","type":"http",
  "response","z":"91f97da0.d49da","name":"","statusCode":"","headers":{"x":690,"y":220,"wires":[]},
{"id":"31048db7.aa84b2","type":"http",
  "in","z":"91f97da0.d49da","name":"","url":"/controlroom","method":"get","upload":false,"swaggerDoc":"","x":200,"y":220,"wires":[["ce6140f0.ee787"]]},
{"id":"ce6140f0.ee787","type":"template",
  "e","z":"91f97da0.d49da","name":"Control Room Web"}
```

```

Page","field":"payload","fieldType":"msg","format":"html","syntax":"mustache","template":"<!
DOCTYPE HTML>\n<html>\n  <head>\n    <title>Sensors and Servos</title>\n    <script
type=\"text/javascript\">\n      var ws;\n      var wsUri = \"ws:\";\n      var loc =
window.location;\n      console.log(loc);\n      if (loc.protocol === \"https:\") { wsUri = \"wss:\";
}\n      // This needs to point to the web socket in the Node-RED flow\n      // ... in this case it's
ws/simple\n      wsUri += \"//\" + loc.host +
loc.pathname.replace(\"controlroom\", \"ws/controlroom\");\n\n      function wsConnect() {\n
console.log(\"connect\",wsUri);\n      ws = new WebSocket(wsUri);\n      //var line = \"\";\n
// either uncomment this for a building list of messages\n      ws.onmessage = function(msg)
{\n      var line = \"\"; // or uncomment this to overwrite the existing message\n      //
parse the incoming message as a JSON object\n      var data = msg.data;\n
//console.log(data);\n      // build the output from the topic and payload parts of the
object\n      line += \"<p>\"+data+\"</p>\";\n      // replace the messages div with the
new \"line\"\n      document.getElementById('messages').innerHTML = line;\n
//ws.send(JSON.stringify({data:data}));\n      }\n      ws.onopen = function() {\n      //
update the status div with the connection status\n
document.getElementById('status').innerHTML = \"connected\";\n      //ws.send(\"Open
for data\");\n      console.log(\"connected\");\n      }\n      ws.onclose = function() {\n
// update the status div with the connection status\n
document.getElementById('status').innerHTML = \"not connected\";\n      // in case of lost
connection tries to reconnect every 3 secs\n      setTimeout(wsConnect,3000);\n      }\n
}\n  \n  function doit(m) {\n      if (ws) { ws.send(m); }\n      }\n      var ssmsg = new
SpeechSynthesisUtterance('Hi there. I am Timo\\'s Watson IoT servoce. Sensor values will be
updated for you once in a minute!');\n      window.speechSynthesis.speak(ssmsg);\n
</script>\n  </head>\n  <body onload=\"wsConnect();\" onunload=\"ws.disconnect();\">\n
<font face=\"Arial\">\n    <h1>Sensors and servos</h1>\n    <p>The sensor locations and
values - updated when a device sends new values:</p>\n    <div id=\"messages\"></div>\n
<button type=\"button\" onclick='doit(\"click\");'>Click to confirm you have got new
values!</button>\n    <br><br>\n    <button type=\"button\"
onclick='doit(\"masterOFF\");'>Turn all servos OFF !</button>\n    <br><br>\n    <button
type=\"button\" onclick='doit(\"masterON\");'>Turn all servos ON !</button>\n    <button
type=\"button\" onclick='doit(\"servo1ON\");'>Only servo 1 ON !</button>\n    <button
type=\"button\" onclick='doit(\"servo2ON\");'>Only servo 2 ON !</button>\n    <hr>\n
<div id=\"status\">unknown</div>\n  </font>\n
</body>\n</html>\n","x":449,"y":220,"wires":[["cf29a616.572df8"]]],{"id":"ecfad89c.fef4a8","ty
pe":"function","z":"91f97da0.d49da","name":"format to a string","func":"msg.payload =
msg.payload.toString();\nreturn
msg;","outputs":1,"noerr":0,"x":450,"y":280,"wires":[["ea9fc0ca.2f179","5422f84e.bcc8a8"]]],{"i
d":"d803ba69.8d6d58","type":"websocket
in","z":"91f97da0.d49da","name":"","server":"9ceb3a2f.eb53e8","client":"","x":470,"y":360,"wir
es":[["b092a074.dd087","6245ffe1.dd6af"]]],{"id":"b092a074.dd087","type":"debug","z":"91f97
da0.d49da","name":"","active":true,"console":"false","complete":"false","x":690,"y":360,"wires
":[],"id":"9569b80d.4440d8","type":"ibmiot
in","z":"91f97da0.d49da","authentication":"boundService","apiKey":"","inputType":"evt","logica
lInterface":"","ruleId":"","deviceId":"MSensors1","applicationId":"","deviceType":"Microcontroll
erSensors","eventType":"+","commandType":"","format":"json","name":"IBM IoT App
In","service":"registered","allDevices":false,"allApplications":false,"allDeviceTypes":false,"allLogi
calInterfaces":false,"allEvents":true,"allCommands":false,"allFormats":false,"qos":"0","x":100,"y

```

[illegible]

```
device","service":"registered","x":760,"y":900,"wires":[{}],{"id":"9ceb3a2f.eb53e8","type":"websocket-listener","z":"91f97da0.d49da","path":"/ws/controlroom","wholemsg":"false"}}
```

Please copy the code either form this document or from a text file:

IBMWatsonSignalsFunctionsServo_Webpage_flow_va.txt

Copy and Paste the code into the NodeRED -editor.

NodeRED-editoriin pitäisi syntyä kuvassa alla näkyvä flow.

A flow in the Node-RED similar to the one seen below should be automatically created with the code.

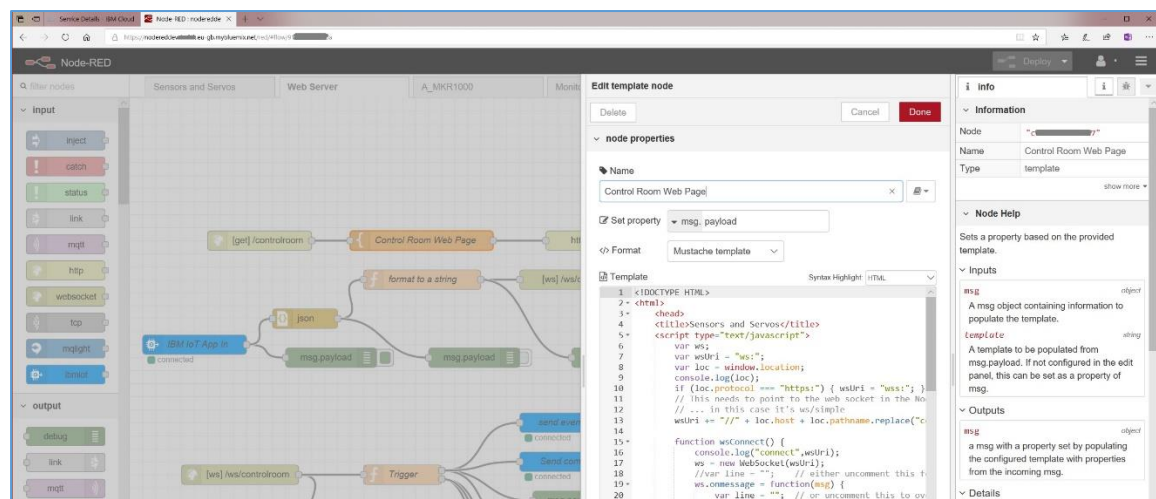


Fig. 4.1 Web Page flow

We will have a closer look at every node in the flow.

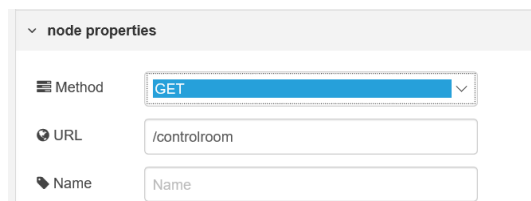


Fig. 4.2 [get]/controlroom

The text “controlroom” will be seen in the address which should be opened with the browser. .
The address is:

<https://<name-of-your-iot-platform>.eu-gb.mybluemix.net/controlroom>

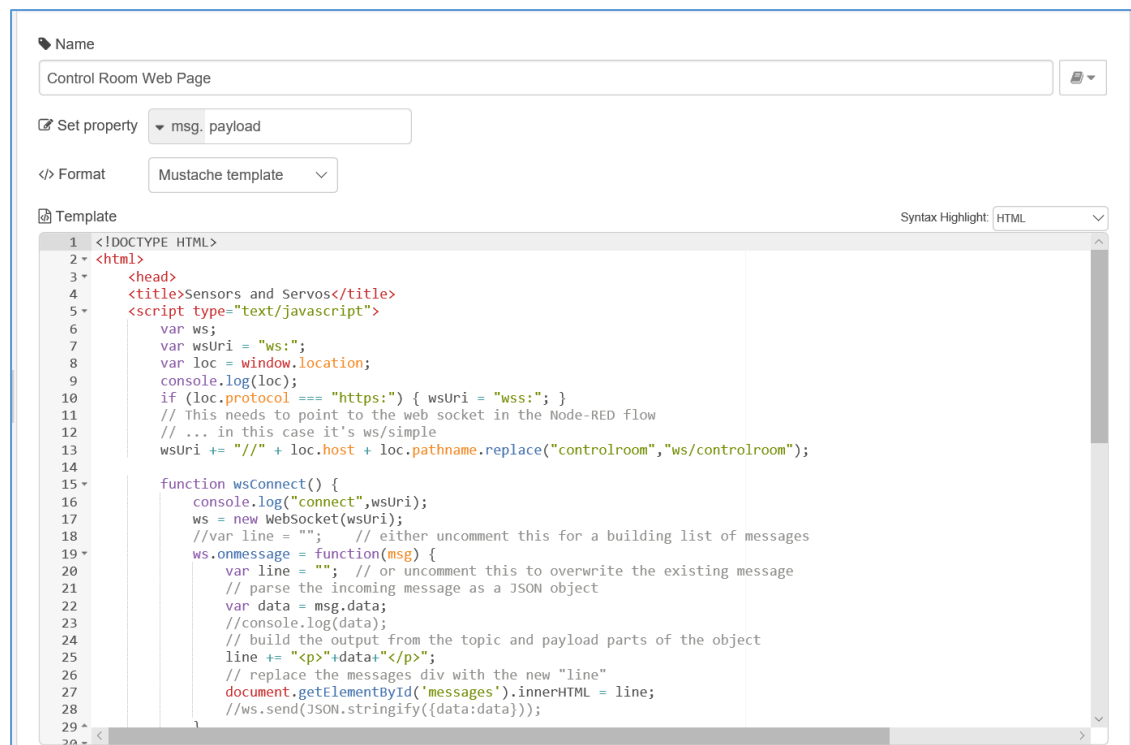


Fig. 4.3 Control Room Web page

The java script code in the node is not seen complete in the picture. The code is copied here as text:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Sensors and Servos</title>
    <script type="text/javascript">
      var ws;
      var wsUri = "ws:";
      var loc = window.location;
      console.log(loc);
      if (loc.protocol === "https:") { wsUri = "wss:"; }
      // This needs to point to the web socket in the Node-RED flow
      // ... in this case it's ws/simple
      wsUri += "://" + loc.host + loc.pathname.replace("controlroom", "ws/controlroom");

      function wsConnect() {
        console.log("connect", wsUri);
        ws = new WebSocket(wsUri);
        //var line = ""; // either uncomment this for a building list of messages
        ws.onmessage = function(msg) {
          var line = ""; // or uncomment this to overwrite the existing message
          // parse the incoming message as a JSON object
          var data = msg.data;
          //console.log(data);
          // build the output from the topic and payload parts of the object
          line += "<p>" + data + "</p>";
          // replace the messages div with the new "line"
          document.getElementById('messages').innerHTML = line;
          //ws.send(JSON.stringify({data:data}));
        }
      }
    </script>
  </head>
  <body>
    <div id="messages"></div>
  </body>
</html>
```

```

ws.onopen = function() {
    // update the status div with the connection status
    document.getElementById('status').innerHTML = "connected";
    //ws.send("Open for data");
    console.log("connected");
}
ws.onclose = function() {
    // update the status div with the connection status
    document.getElementById('status').innerHTML = "not connected";
    // in case of lost connection tries to reconnect every 3 secs
    setTimeout(wsConnect,3000);
}
}

function doit(m) {
    if (ws) { ws.send(m); }
}
var ssmg = new SpeechSynthesisUtterance('Hi there. I am Timo\'s Watson IoT servoce. Sensor values will be
updated for you once in a minute!');
window.speechSynthesis.speak(ssmsg);
</script>
</head>
<body onload="wsConnect();" onunload="ws.disconnect();">
    <font face="Arial">
    <h1>Sensors and servos</h1>
    <p>The sensor locations and values - updated when a device sends new values:</p>
    <div id="messages"></div>
    <button type="button" onclick='doit("click");'>Click to confirm you have got new values!</button>
    <br><br>
    <button type="button" onclick='doit("masterOFF");'>Turn all servos OFF !</button>
    <br><br>
    <button type="button" onclick='doit("masterON");'>Turn all servos ON !</button>
    <button type="button" onclick='doit("servo1ON");'>Only servo 1 ON !</button>
    <button type="button" onclick='doit("servo2ON");'>Only servo 2 ON !</button>
    <hr/>
    <div id="status">unknown</div>
    </font>
</body>
</html>

```

Code 4.1 html code for the web page

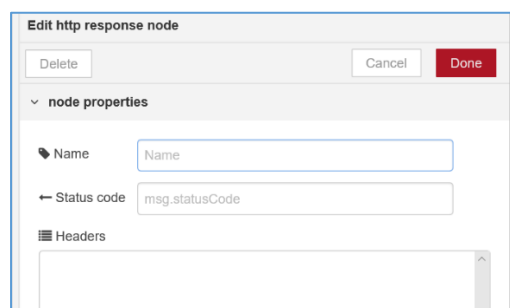


Fig. 4.4 The http response node will be left empty

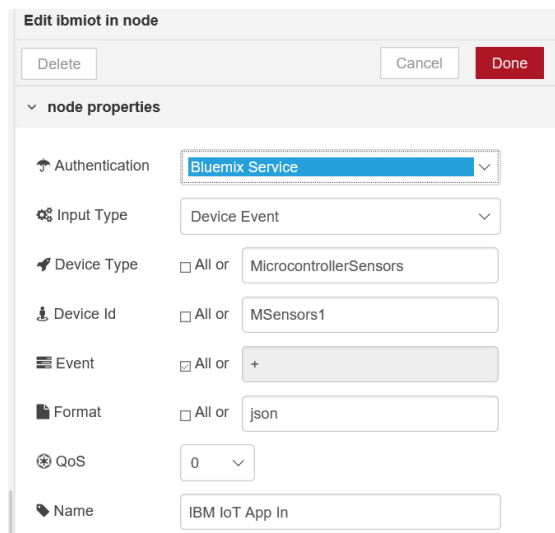


Figure 4.5 shows the configuration interface for the 'ibmiot' node in Node-RED. The 'Authentication' is set to 'Bluemix Service'. The 'Input Type' is 'Device Event'. The 'Device Type' is 'All or MicrocontrollerSensors'. The 'Device Id' is 'All or MSensors1'. The 'Event' is 'All or +'. The 'Format' is 'All or json'. The 'QoS' is '0'. The 'Name' is 'IBM IoT App In'.

Fig. 4.5 ibm iot node

Input node of type ibm iot receives messages from Watson IoT Platform. It is possible to define precisely which messages will be read. It is possible to read all types of messages from all devices – simply select all in every option. But there is a risk. If you will select all as message type and all as device it is possible to get a message making an endless loop!

The node JSON converts between JSON strings and JSON objects. There is no need to change anything.

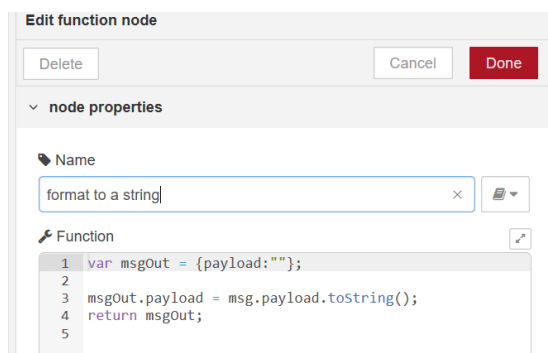


Figure 4.6 shows the configuration interface for the 'function' node in Node-RED. The 'Name' is 'format to a string'. The 'Function' is a JavaScript code snippet:

```
1 var msgOut = {payload:''};
2 msgOut.payload = msg.payload.toString();
3 return msgOut;
4
5
```

Fig 4.6 function node

The function node is either picking up the whole content of the payload or it can be defined to pick up only certain objects from a JSON formatted message payload.

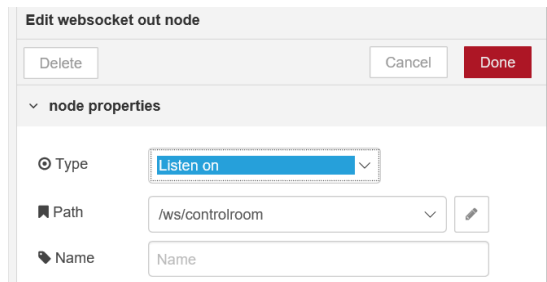


Fig. 4.7 Output node Websocket

In the Web socket -node a name for the web socket is defined. Of course this has to be the same as used in the java script code.

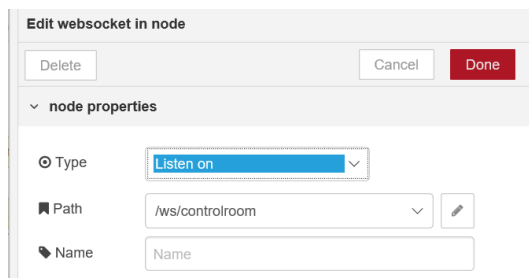


Fig. 4.8 Input node Web socket

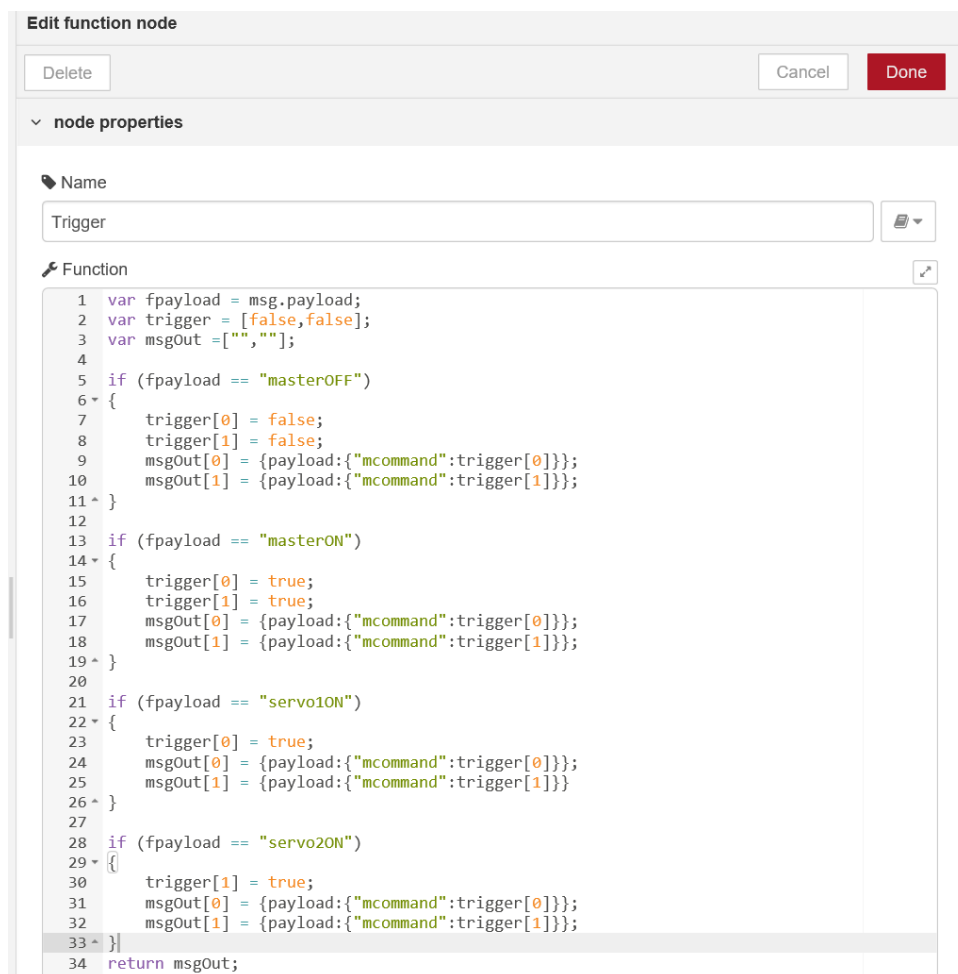


Fig. 4.9 Function node Trigger

The Java script code for the function node can be copied here as text.

```
var fpayload = msg.payload;
var trigger = [false,false];
var msgOut=["",""];

if (fpayload == "masterOFF")
{
    trigger[0] = false;
    trigger[1] = false;
    msgOut[0] = {payload:"mcommand":trigger[0]};
    msgOut[1] = {payload:"mcommand":trigger[1]};
}

if (fpayload == "masterON")
{
    trigger[0] = true;
    trigger[1] = true;
    msgOut[0] = {payload:"mcommand":trigger[0]};
    msgOut[1] = {payload:"mcommand":trigger[1]};
}

if (fpayload == "servo1ON")
{
    trigger[0] = true;
    msgOut[0] = {payload:"mcommand":trigger[0]};
    msgOut[1] = {payload:"mcommand":trigger[1]}
}

if (fpayload == "servo2ON")
{
    trigger[1] = true;
    msgOut[0] = {payload:"mcommand":trigger[0]};
    msgOut[1] = {payload:"mcommand":trigger[1]};
}
return msgOut;
```

Code 4.2 Java script code in the node which will make the decisions.

Edit ibmiot out node

Delete Cancel Done

node properties

Authentication: Bluemix Service

Output Type: Device Event

Device Type: MicrocontrollerServos

Device Id: MServos1

Event Type: start

Format: json

Data: no command sent

QoS: 0

Name: send event to IBM IoT PI

Fig. 4.10 Output node ibm iot – Device Event

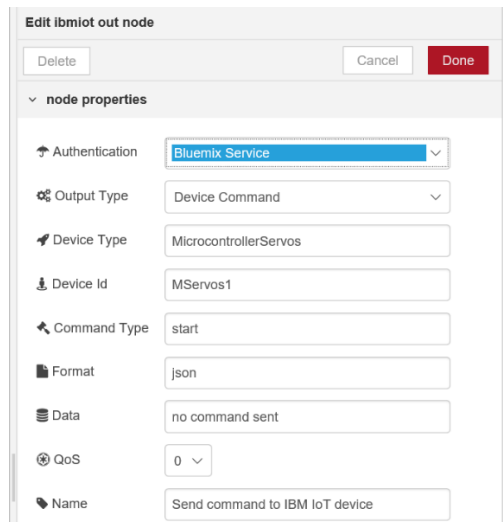


Fig. 4.11 Output node ibm iot – Device Command

Please note! It is possible to send to the Watson IoT Platform messages with type event and messages with type command.

We have now gone through all the nodes in the Node-RED flow.

5. A Commissioning task for a user interface created as a Web Page

Assignment 3

Test the operation of the web page in the given example.

In the java script code in the web page node there are instruction on comment lines. The instruction tell how it is possible to get the events visible as line after line in the web page. Try this!

In the report: Does the code operate as told. Why is it operating as told? Why it is not operating as told?