1. What is in the box?

eIoT3500 contains following items

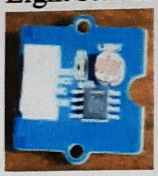
1. Intel Galileo gateway platform
    a. Enclosed in casing for protection and ease of use
    b. 8GB micro SD card (pre-flashed with Linux BSP)
    c. Integrated wifi-BT mini PCIe card with Antenna connector mounted
2. Accessories box
    a. Leoxsys Wireless Nano Adaptor
    b. USB to Serial FTDI cable
    c. 32GB Pendrive pre-flashed with all necessary software
    d. Antenna (2 Nos) for wifi-BT
    e. Power adapter with multi-geo plug options
3. Crossover Ethernet cable
4. Sensor box
    a. 16x2 LCD RGB backlight

    b. Sound sensor

    c. LED socket kit (Red , Green , Blue)

    d. Touch sensor

e. Light sensor



f. Buzzer

g. Rotary angle sensor

h. Button

i. Temperature sensor

j. Relay

k. Servo motor with accessories

l. Connecting wires – 10 Nos.

2. Connecting the gateway platform hardware

1. Locate Base Shield in the sensor box , below LCD Module
2. Insert the Base Shield on the Galileo board
   Note: Ensure that all the pins are correctly matched before inserting the base shield
   Note: Make sure that the switch on Base Shield is set to 5V before powering on the board
3. Connect Antenna to the Galileo board
   Note: Antenna should be connected before powering on the board. Otherwise wifi will not work
4. Connect FTDI cable to the Galileo board

   Note: Match the pin1 on the board with black wire (denoted by
   Hold the FTDI cable by plastic casing. Do not attempt to pull cable by the wires directly

3. Booting the Pen Drive on your Host

1. Locate the 32GB SanDisk bootable pen drive inside the accessories box. This pen drive is loaded with a custom operating system with all the necessary software tools pre-installed to kick start the application development with eIoT3500.
2. You need a host machine a.k.a computer to boot the pre-installed pen drive.
   Note: Please ensure that your host machinea.k.a computer has a 64 bit processor architecture before you attempt to boot the pre-installed pen drive provided.
   Check the PROCESSOR_ARCHITECTURE environment variable on your host a.ka. Computer to know whether the processor architecture is 64 bit or 32 bit.
   Open command line in your windows PC and type
   echo PROCESSOR_ARCHITECTURE.
   If the output gives AMD64 then your processor is 64 bit else the output will be x86 which tells us that the processor architecture is 32-bit.
   If your host machine a.k.a computer is already running any flavor of Linux, then please follow the instructions provided in the link below.
   http://www.cyberciti.biz/faq/linux-how-to-find-if-processor-is-64-bit-or-not/
3. Shutdown your host machine a.k.a computer
4. Connect the USB bootable pen drive to your host machine a.k.a computer USB port.
5. Power up the system and enter the BIOS set up menu.
   Note: Please check how to enter BIOS set up menu on your host machine. Usually pressing the keys F1/F2 immediately after pressing power button will allow you to enter the BIOS set up menu option.
6. Navigate through the BIOS set up menu and disable the secure boot option if present.
7. Navigate to the boot order option in the BIOS set up menu and move the USB pen drive to the top of the list.
8. After all the above changes are done, save the BIOS settings and boot the setting. Usually pressing F10 key will allow you to save and exit from the BIOS set up menu environment.
9. After all the above steps are followed the system should automatically boot into the Ubuntu OS pre-loaded in the pen drive.

4. Connecting the host to a wifi access point

1. Boot the host machine a.k.a computer from the pen drive. Refer to section 3.0 for more details on how to boot from pen drive.

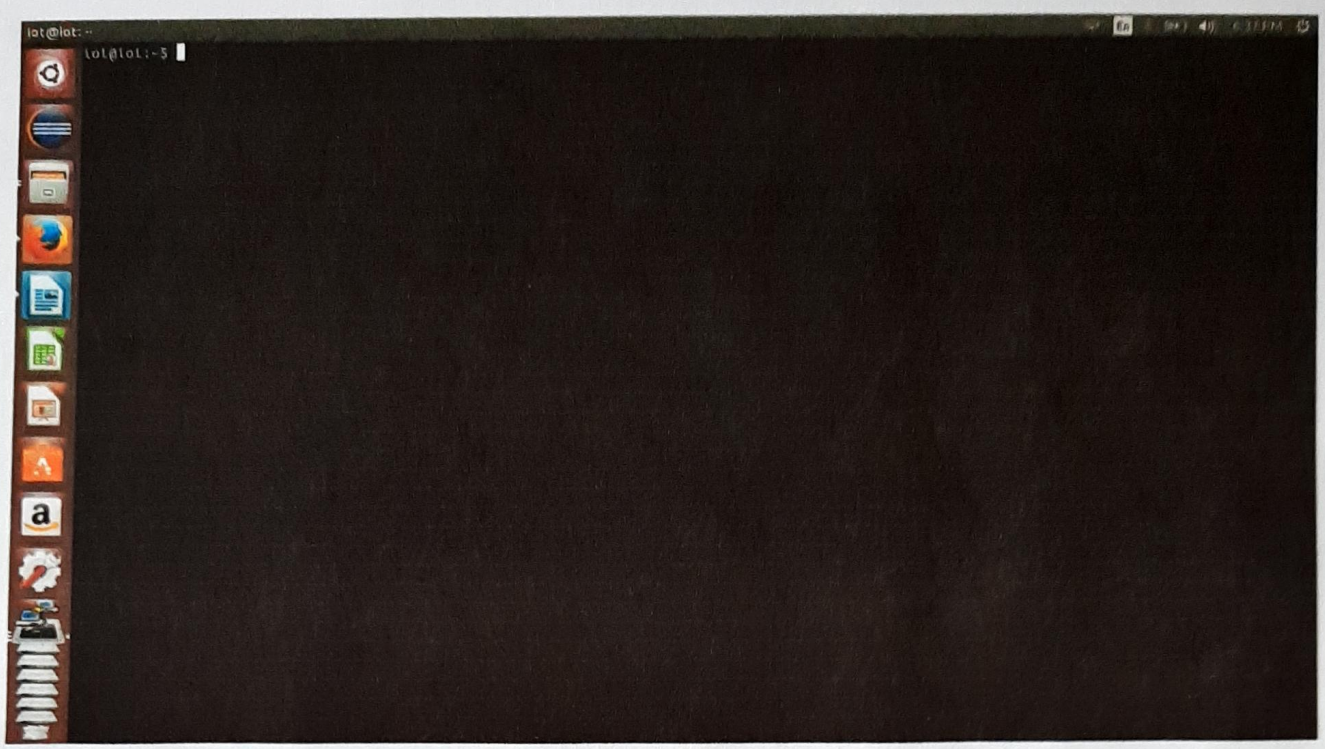
2. One the host machine is booted into the pen drive environment, check if your wifi card is detected. If yes then connect the host wifi to a wifiAcessPoint.
   Note: Use the same wifi Access Point to connect the gateway target as well.Please refer to section 5.0 to connect gateway platform to a wifi access point.

3. In case your host machine does not have an inbuilt wifiadapter,simply plug in the LeoxsysWiFi dongle to the host machine. This will add WiFi capability to your host machine.
   Note: The LeoxsysWiFi adapter can be located in the accessories box.

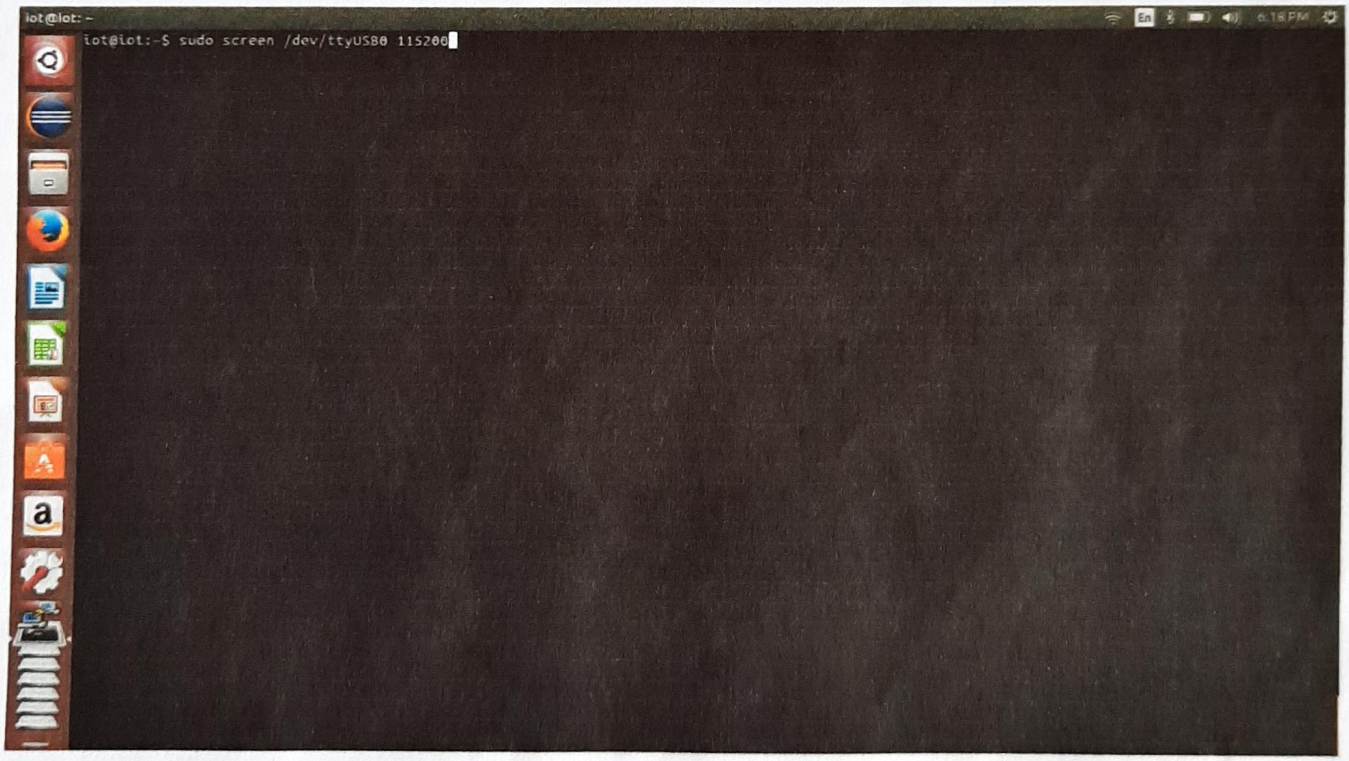
4. Iftheinbuilt wifi adapter of your host machine fails to detect in the pen drive environment, locate the Leoxsys adapter and plug it in on one of the USB ports of your host machine. This will allow your host machine to connect wifi using the wifi dongle provided with the kit.

5. Connecting the gateway to a wifi access point

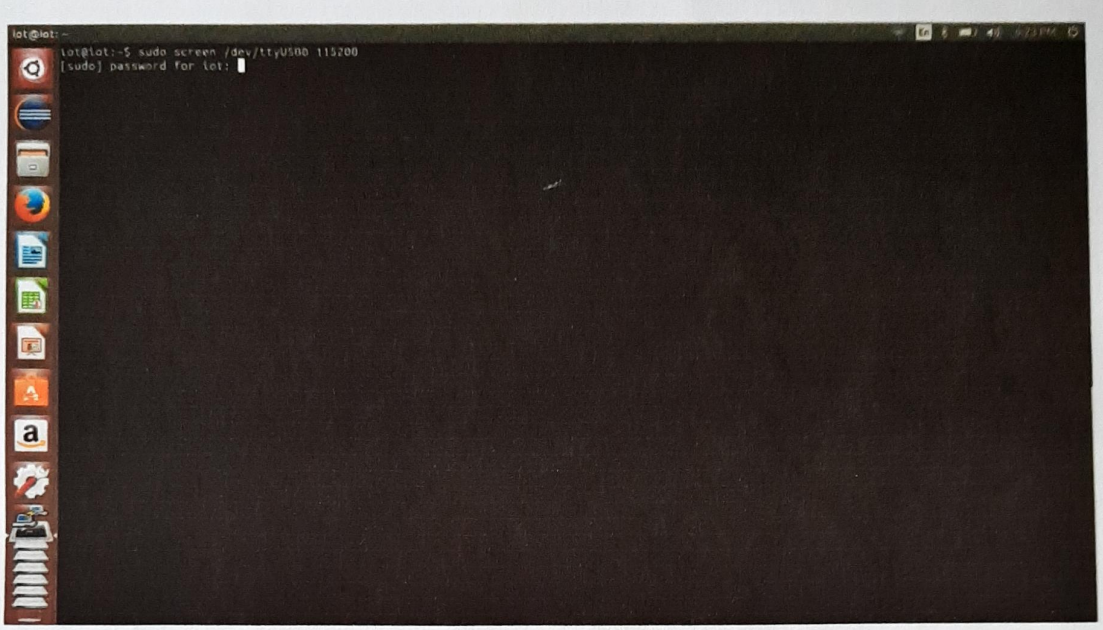
1. Make sure that the target is completely powered off before performing the steps described below.
2. Also make sure that the antennas are connected on the gateway platform before performing the steps mentioned below
3. Boot the host machine a.k.a computer from the pen drive. Refer to section 3.0 for more details on how to boot from pen drive.
4. Connect the FTDI to USB converter cable to one of the USB ports on your host machine.
5. Open a terminal on the host machine as shown in the screen shot below.



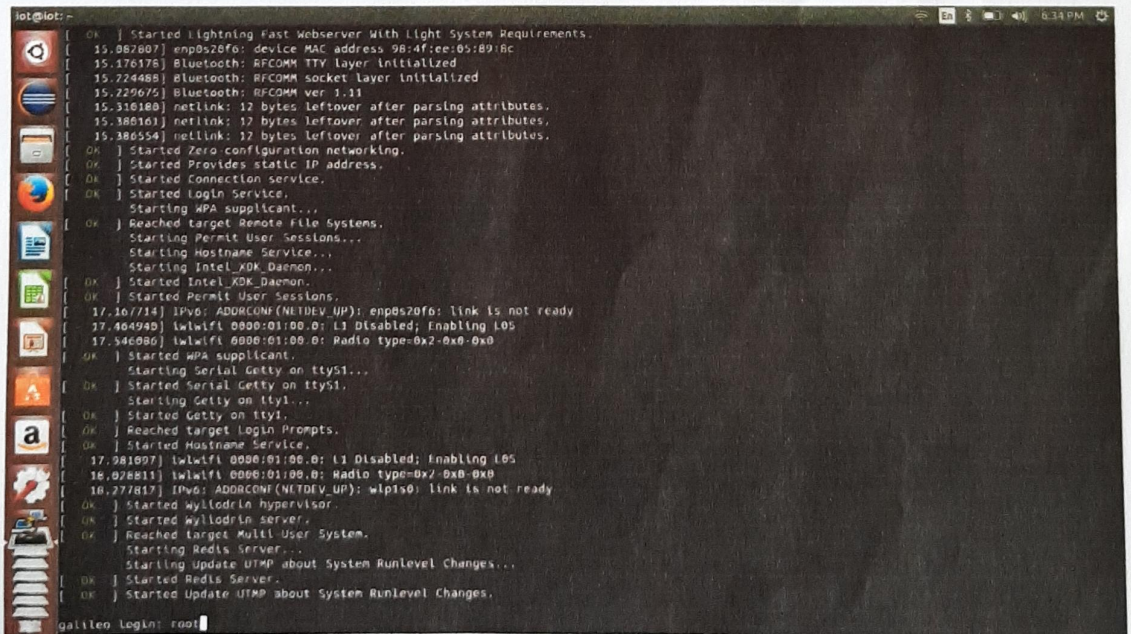
6. Enter the command **sudo screen /dev/ttyUSB0 115200** in the terminal.



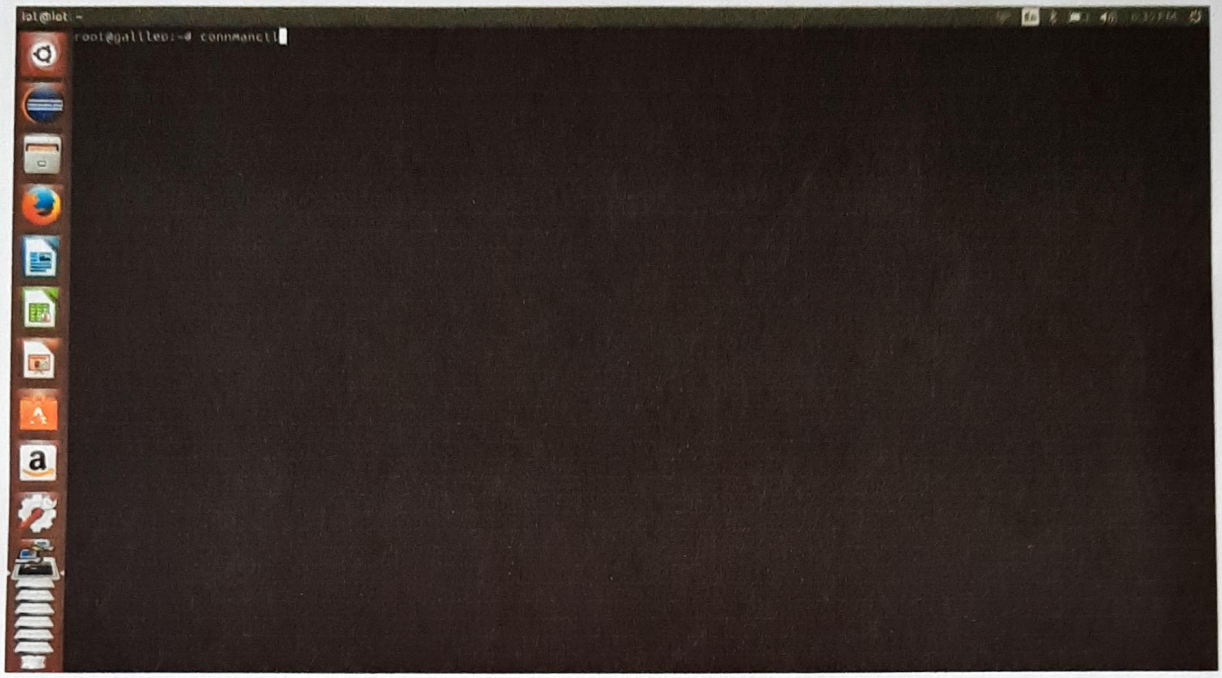
7. You will be prompted to enter the password. Enter **iot123** as the password.



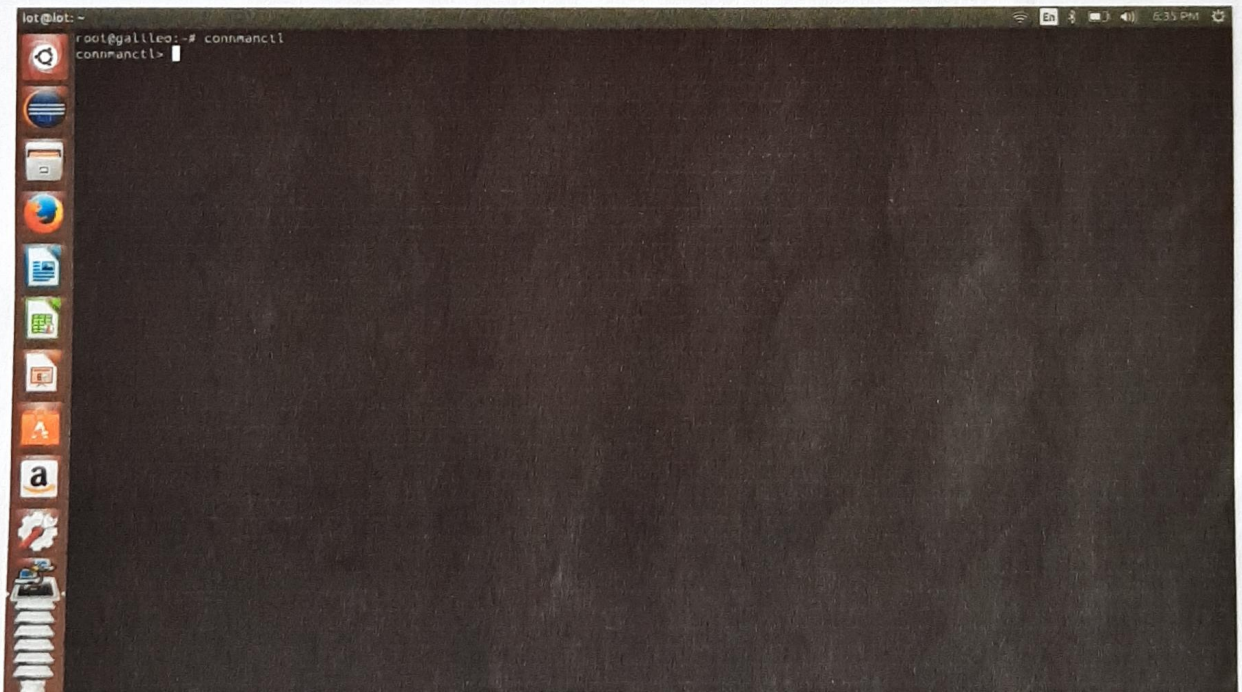
8. Power on the gateway platform using the AC adapter provided. The green LED on the board lights up indicating that the gateway platform is powered up correctly.
9. The gateway platform starts to boot a yocto linux image which is flashed on the micro SD of the platform. The screen console which was opened earlier will start to display the booting process of the gateway platform.
10. Wait until we see the follow on the screen window. This indicates that you need to login into the target. Please type root as shown below and press enter
    Galileo: root



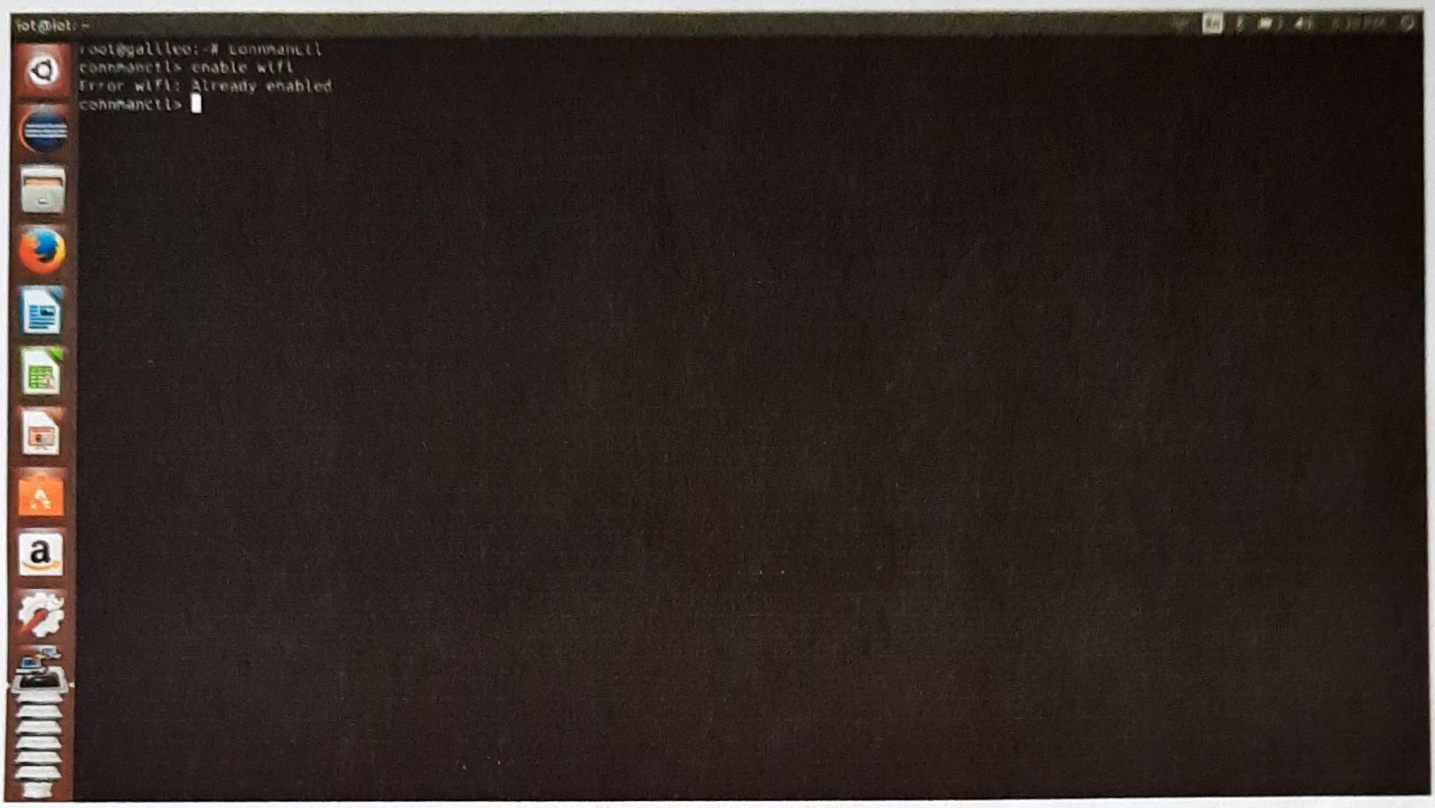
11. Now we should configure and connect the gateway wireless adapter to a wifi access point. We will use the widely used configuration tool called connman.
12. At the terminal enter **connmanctl**



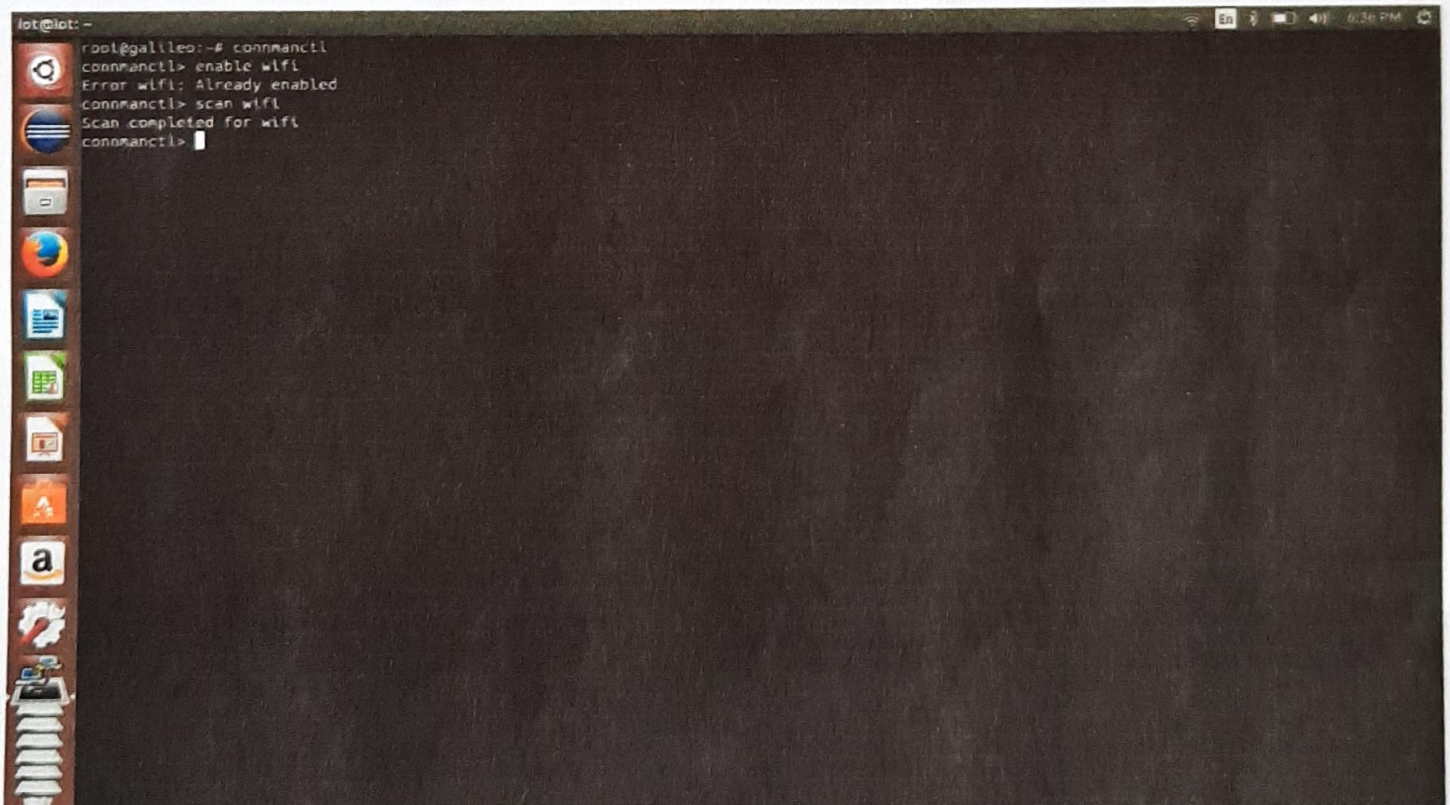
Next type the commands at the connmanctl console



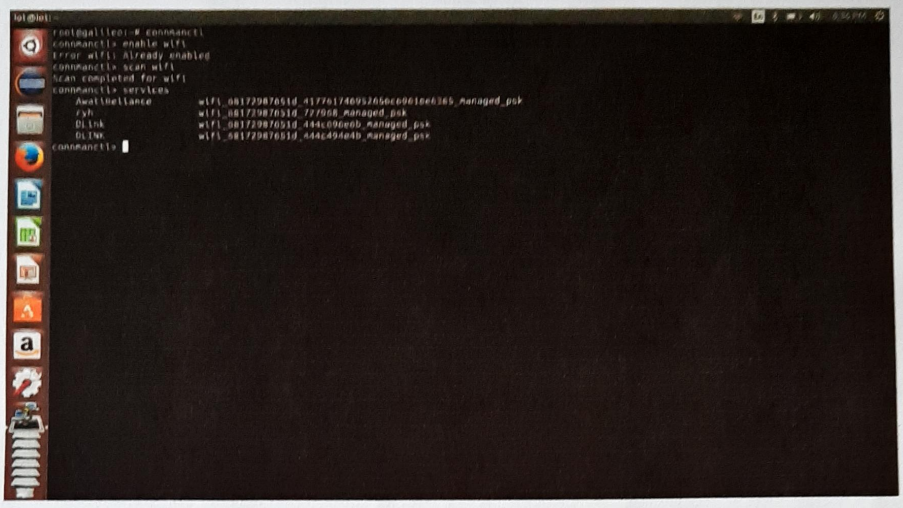
enable wifi



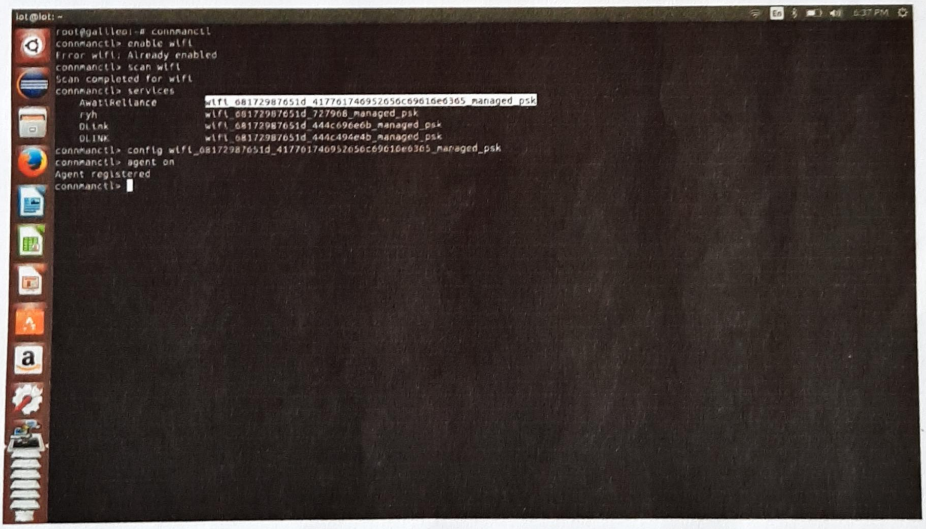
scan wifi



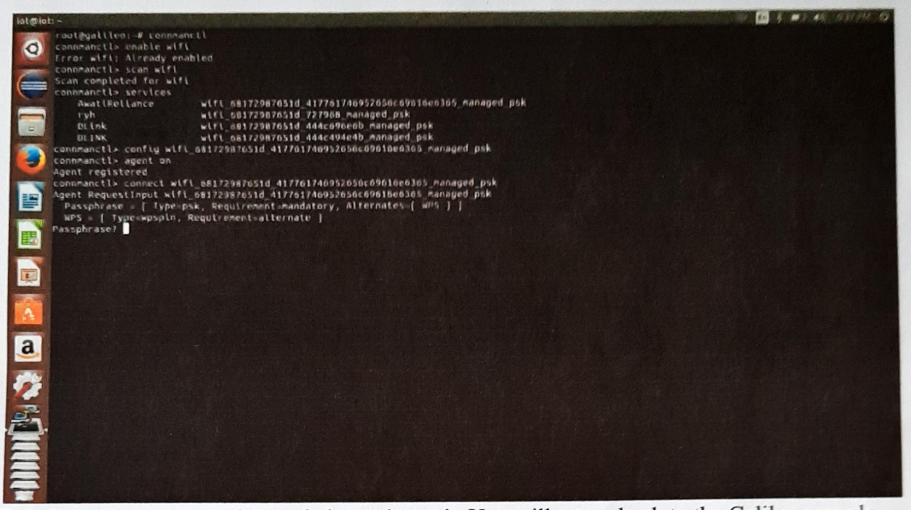
services



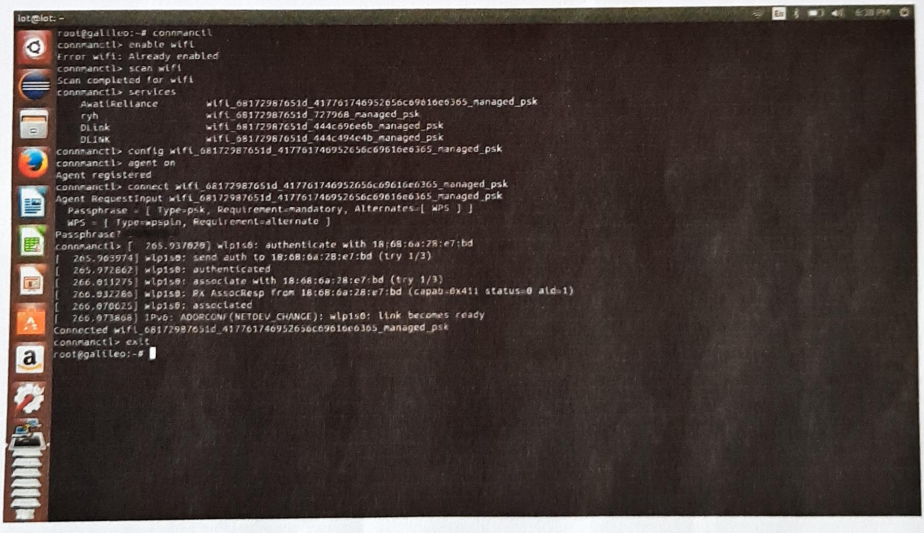
After you type the services command the screen will show the list of access points scanned by the gateway. You should see the name and ssid of the wifi access point which you need to connect to. Copy the SSID from the terminal and type the following commands



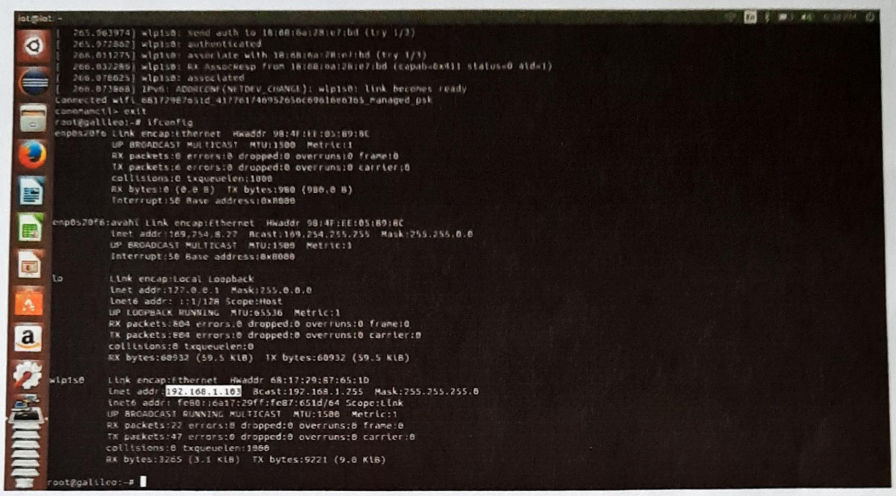
config (paste the SSID of the wifi AP)
agent on
connect (paste the SSID of the wifi AP)



Exit from the connmanctl console by typing exit. You will return back to the Galileo console.



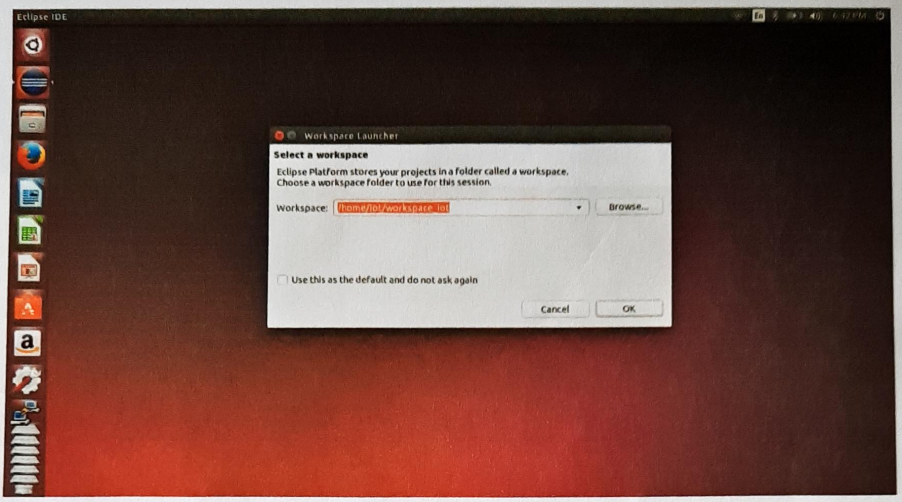
Type the command **ifconfig** at the Galileo console to make sure that the target gateway platform is connected to the wifi AP.
Please make a note of the IP address assigned to your target gateway platform.

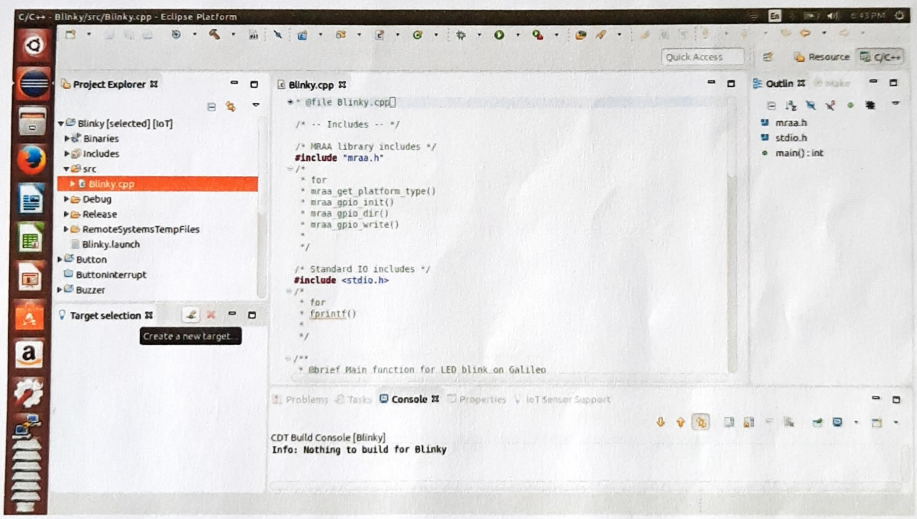


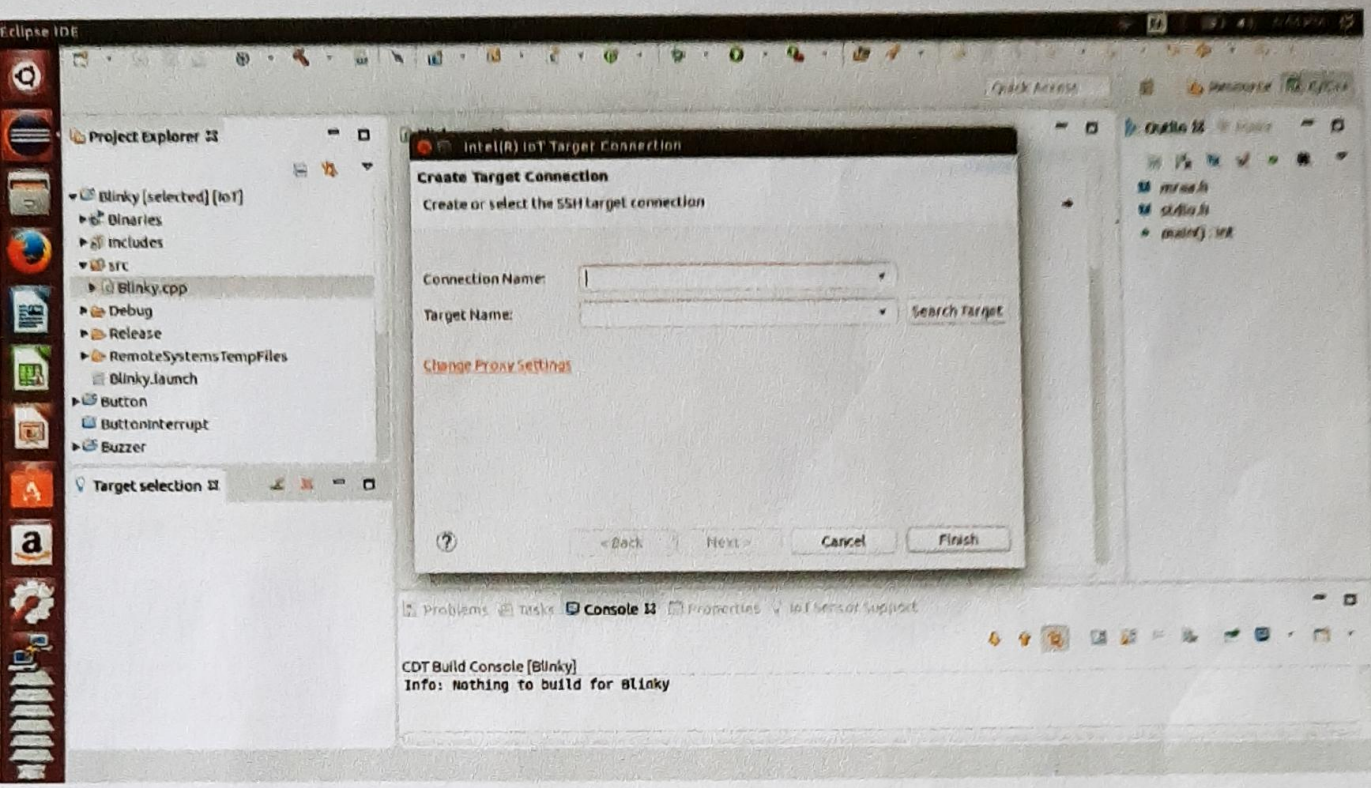
Exit the screen console by disconnecting the FTDI to USB cable.

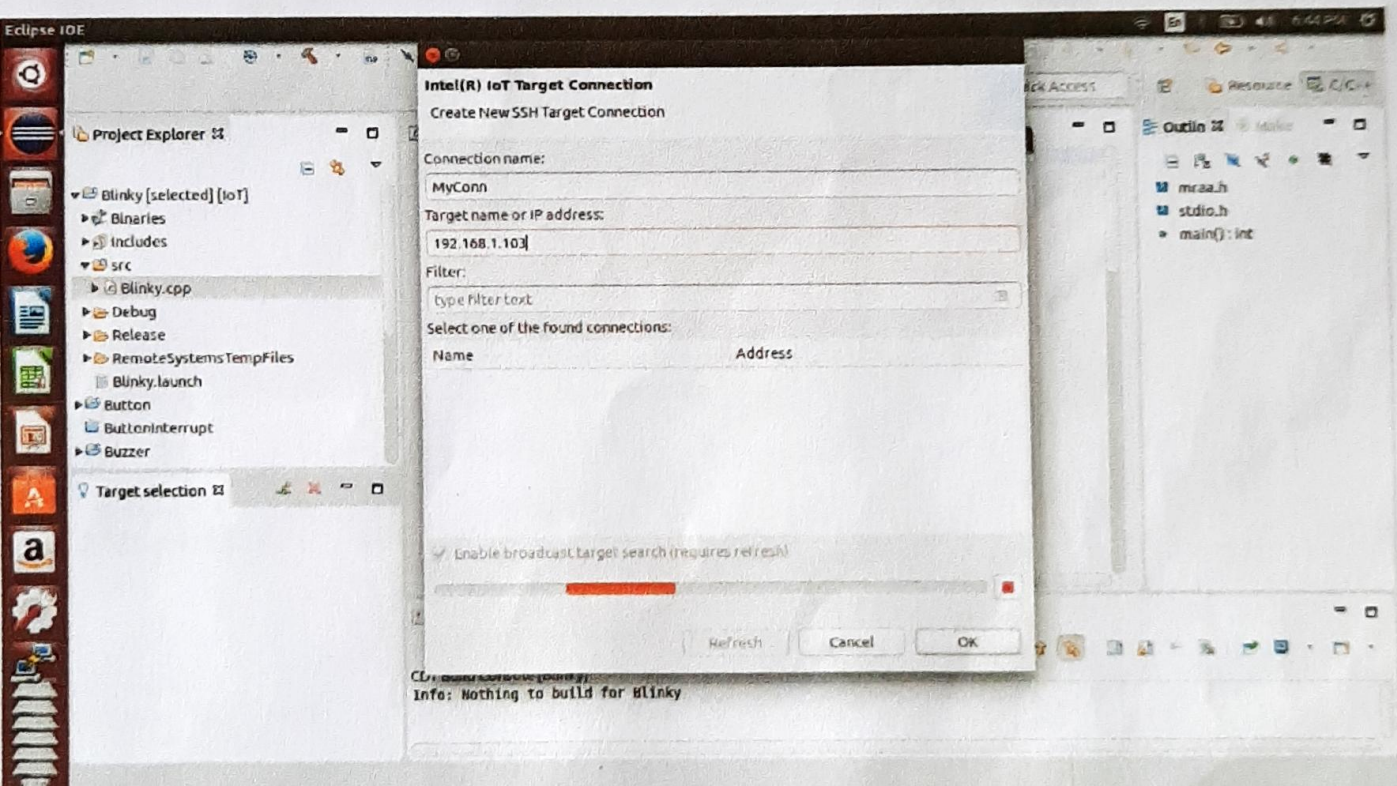
6. Getting started with the exercises

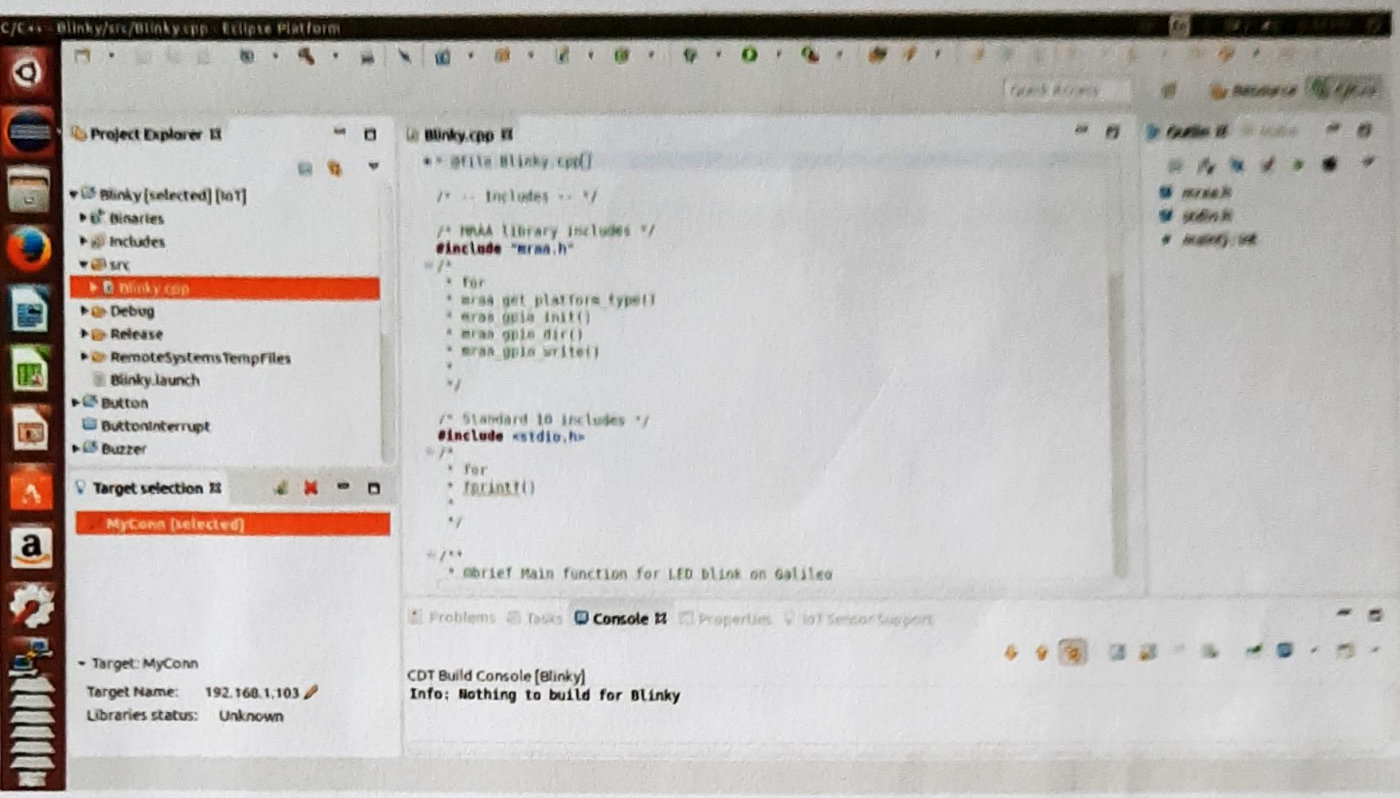
a) Launching the Eclipse IDE
b) Opening a project and building
c) Configuring the target in the Eclipse IDE
d) Loading the project on the target and running
e) Building the project and debugging the same
f) Setting breakpoints and single stepping through the code.

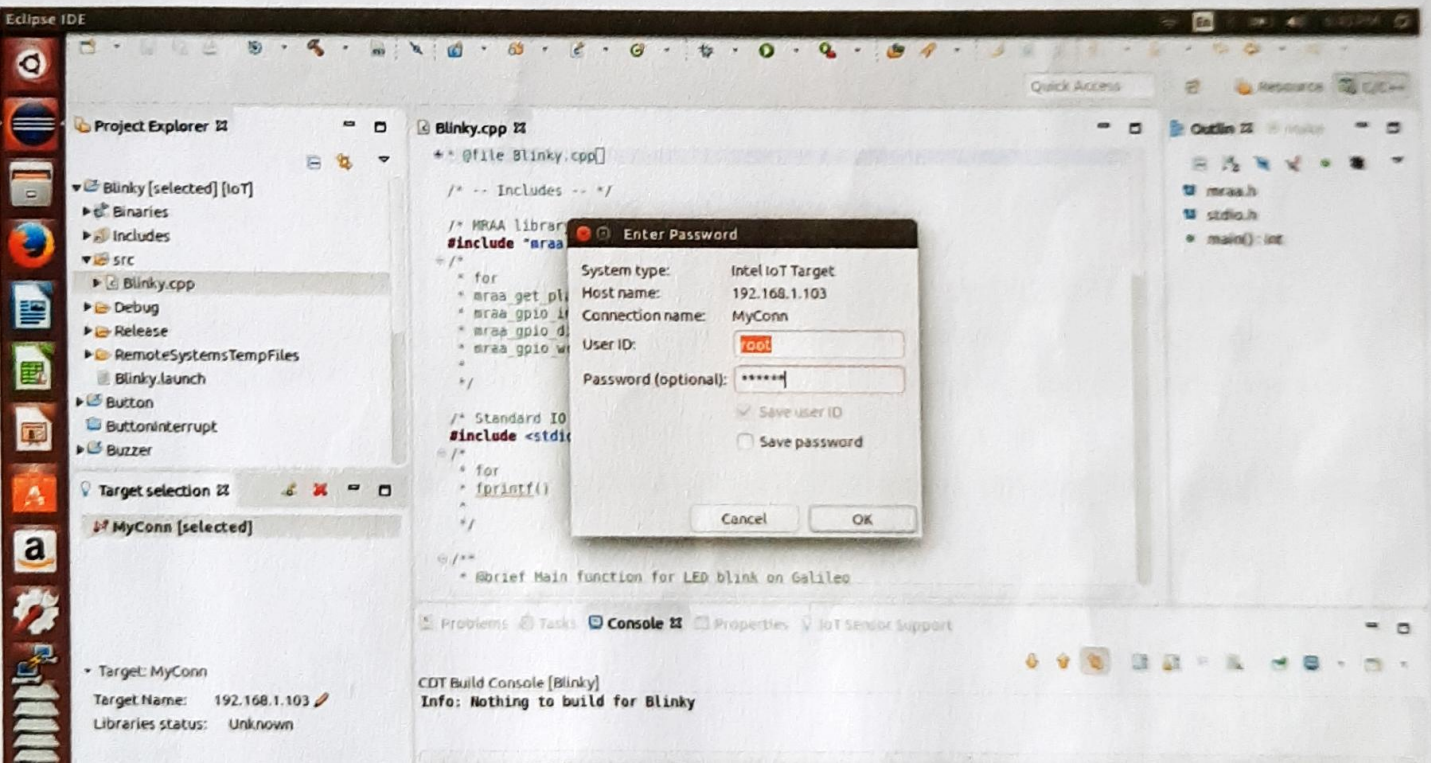





Eclipse IDE
Project Explorer
Blinky [selected] [IoT]
Binaries
Includes
src
Blinky.cpp
Debug
Release
RemoteSystemsTempFiles
Blinky.launch
Button
ButtonInterrupt
Buzzer
Target selection
Intel(R) IoT Target Connection
Create Target Connection
Create or select the SSH target connection
Connection Name:
Target Name:
Search Target
Change Proxy Settings
< Back
Next >
Cancel
Finish
Problems
Tasks
Console
Properties
CDT Build Console [Blinky]
Info: Nothing to build for Blinky

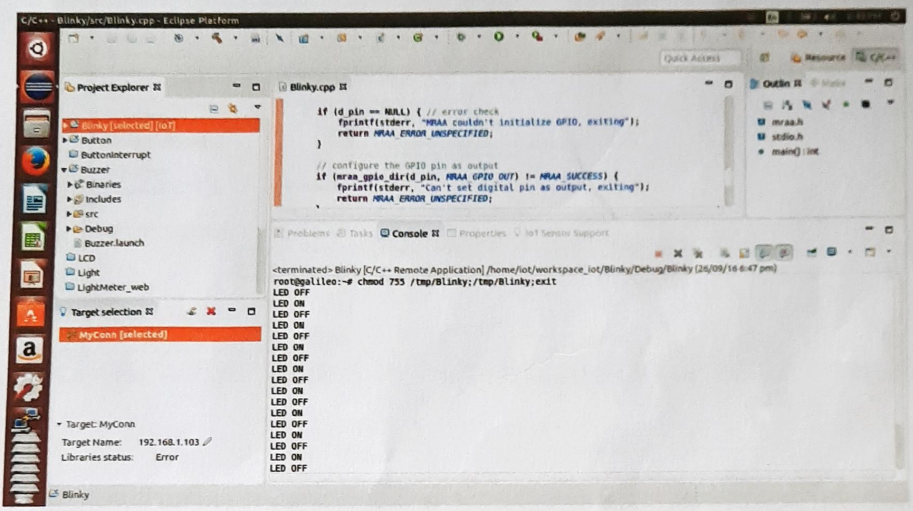

Eclipse IDE
Project Explorer
Blinky [selected] [IoT]
Binaries
Includes
src
Blinky.cpp
Debug
Release
RemoteSystemsTempFiles
Blinky.launch
Button
ButtonInterrupt
Buzzer
Target selection
Intel(R) IoT Target Connection
Create New SSH Target Connection
Connection name:
MyConn
Target name or IP address:
192.168.1.103
Filter:
type filter text
Select one of the found connections:
Name
Address
Enable broadcast target search (requires refresh)
Refresh
Cancel
OK
Outline
mraa.h
stdio.h
main() : int
Info: Nothing to build for Blinky


C/C++ - Blinky/src/Blinky.cpp - Eclipse Platform
Project Explorer
Blinky [selected] [IoT]
Binaries
Includes
src
Blinky.cpp
Debug
Release
RemoteSystemsTempFiles
Blinky.launch
Button
ButtonInterrupt
Buzzer
Target selection
MyConn [selected]
Target: MyConn
Target Name: 192.168.1.103
Libraries status: Unknown
Blinky.cpp
/* -- Includes -- */
/* MRAA library includes */
#include "mraa.h"
/*
* for
* mraa_get_platform_type()
* mraa_gpio_init()
* mraa_gpio_dir()
* mraa_gpio_write()
*
*/
/* Standard IO includes */
#include <stdio.h>
/*
* for
* fprintf()
*
*/
/**
* @brief Main function for LED blink on Galileo
Problems
Tasks
Console
Properties
IoT Sensor Support
CDT Build Console [Blinky]
Info: Nothing to build for Blinky
Outline
mraa.h
stdio.h
main() : int


Eclipse IDE
Quick Access
Project Explorer
Blinky [selected] [IoT]
Binaries
Includes
src
Blinky.cpp
Debug
Release
RemoteSystemsTempFiles
Blinky.launch
Button
ButtonInterrupt
Buzzer
Target selection
MyConn [selected]
Target: MyConn
Target Name: 192.168.1.103
Libraries status: Unknown
Blinky.cpp
/* -- Includes -- */
/* for
* fprintf()
*/
/**
* @brief Main function for LED blink on Galileo
Enter Password
System type: Intel IoT Target
Host name: 192.168.1.103
Connection name: MyConn
User ID: root
Password (optional): ******
Save user ID
Save password
Cancel
OK
Problems
Tasks
Console
Properties
IoT Sensor Support
CDT Build Console [Blinky]
Info: Nothing to build for Blinky
Outline
mraa.h
stdio.h
main() : int

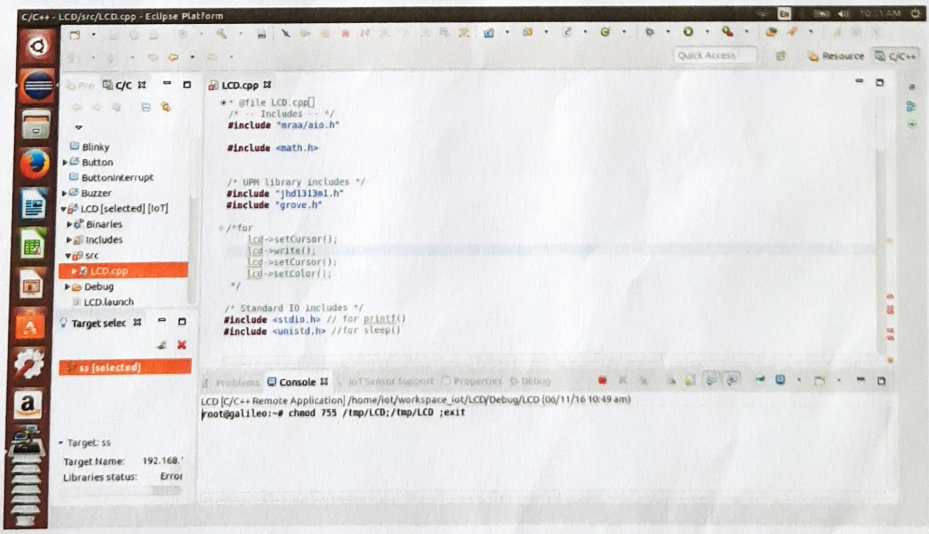
- Blinky Program:

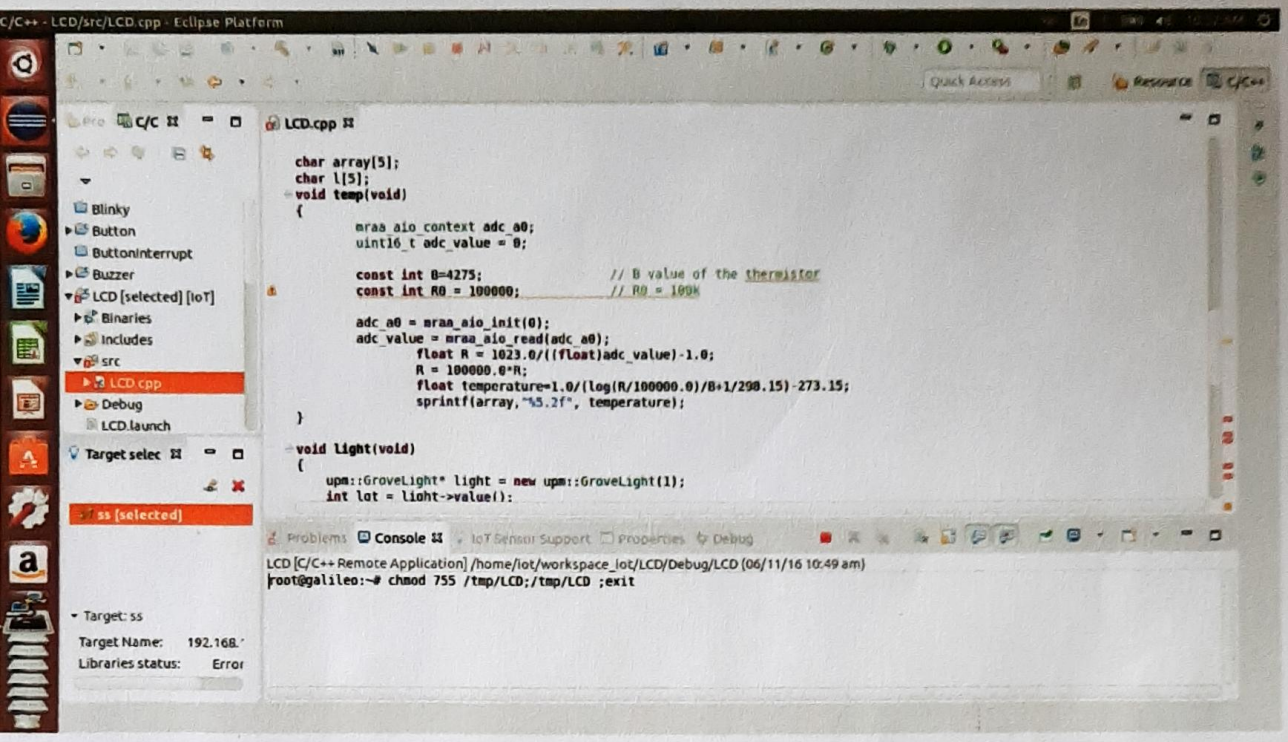
Once we run the program the LED on the Galileo board goes on and off depending on iterations defined in the code.

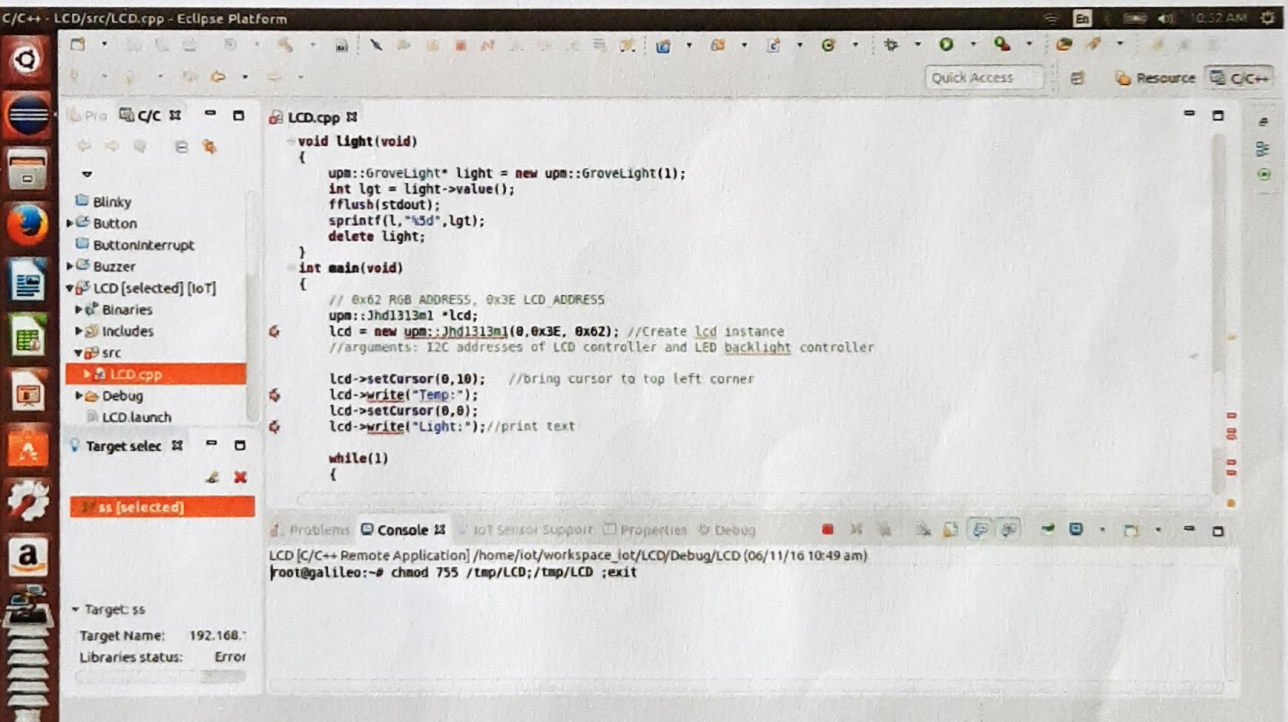


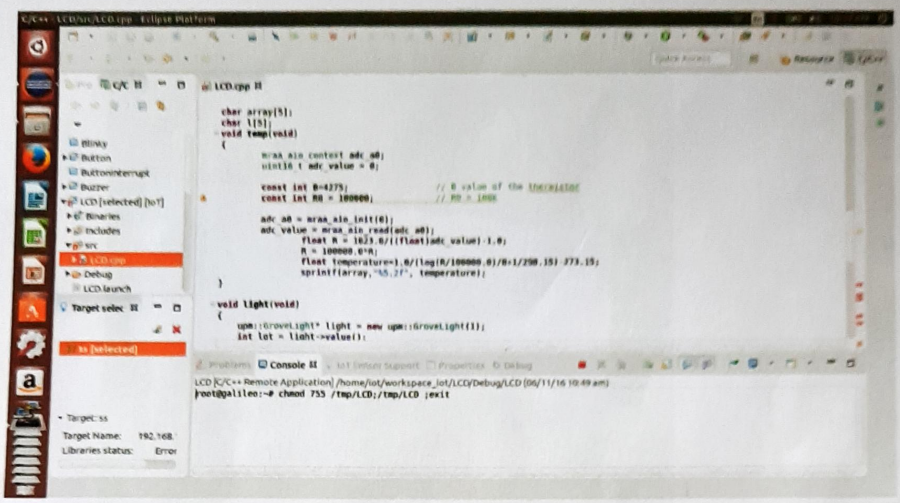
- Integrated Sensors Program

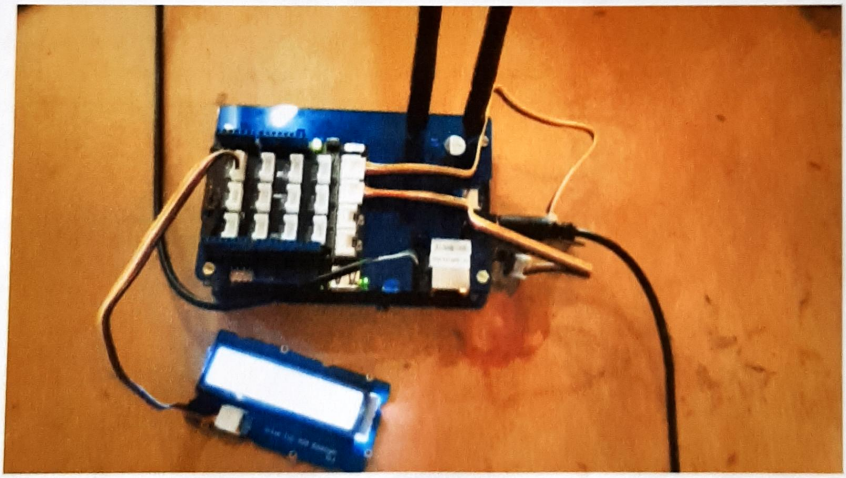
So far we have worked on individual sensors. In the following project we have clubbed three sensor namely Temperature sensor, Light sensor and LCD. The temperature sensor and light sensor continuously monitors the temperature value and the light intensity and displays it on the LCD screen. The code is as shown below in the following images.

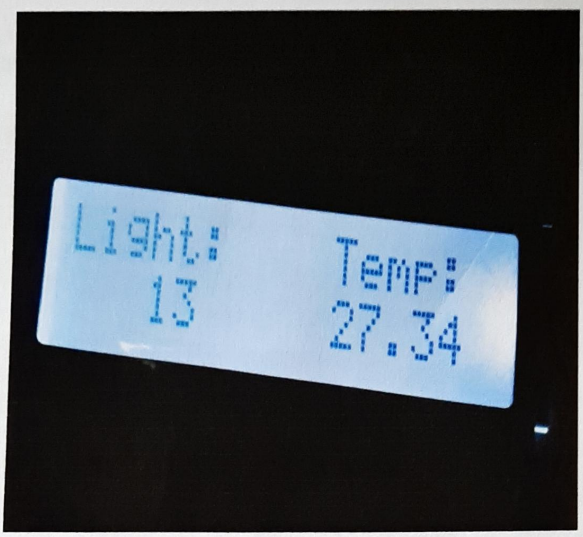



C/C++ - LCD/src/LCD.cpp - Eclipse Platform
LCD.cpp
char array[5];
char l[5];
void temp(void)
{
mraa_aio_context adc_a0;
uint16_t adc_value = 0;
const int B=4275; // B value of the thermistor
const int R0 = 100000; // R0 = 100k
adc_a0 = mraa_aio_init(0);
adc_value = mraa_aio_read(adc_a0);
float R = 1023.0/((float)adc_value)-1.0;
R = 100000.0*R;
float temperature=1.0/(log(R/100000.0)/B+1/298.15)-273.15;
sprintf(array,"%5.2f", temperature);
}
void light(void)
{
upm::GroveLight* light = new upm::GroveLight(1);
int lgt = light->value();
Blinky
Button
ButtonInterrupt
Buzzer
LCD [selected] [IoT]
Binaries
Includes
src
LCD.cpp
Debug
LCD.launch
Problems
Console
IoT Sensor Support
Properties
Debug
LCD [C/C++ Remote Application] /home/iot/workspace_iot/LCD/Debug/LCD (06/11/16 10:49 am)
root@galileo:~# chmod 755 /tmp/LCD;/tmp/LCD ;exit
Target: ss
Target Name: 192.168.
Libraries status: Error

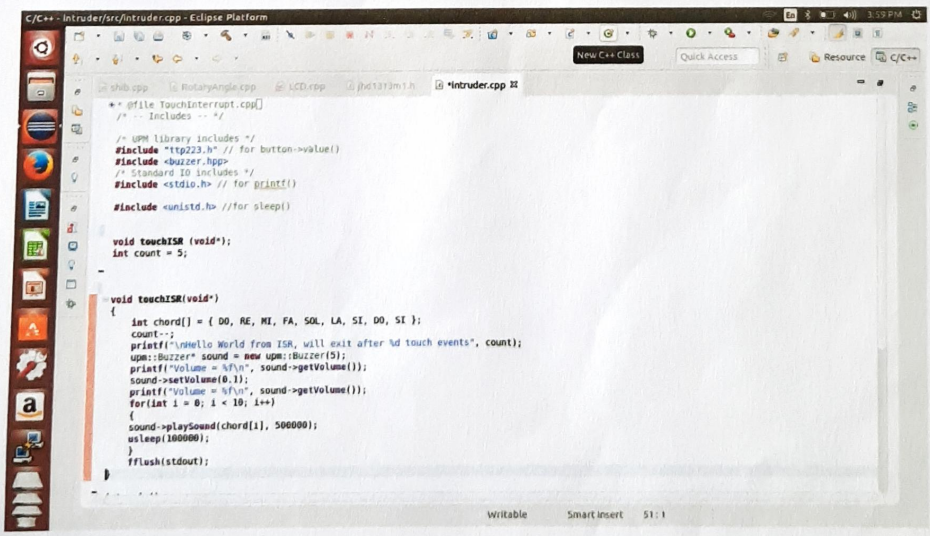

C/C++ - LCD/src/LCD.cpp - Eclipse Platform
LCD.cpp
void light(void)
{
upm::GroveLight* light = new upm::GroveLight(1);
int lgt = light->value();
fflush(stdout);
sprintf(l,"%5d",lgt);
delete light;
}
int main(void)
{
// 0x62 RGB_ADDRESS, 0x3E LCD_ADDRESS
upm::Jhd1313m1 *lcd;
lcd = new upm::Jhd1313m1(0,0x3E, 0x62); //Create lcd instance
//arguments: I2C addresses of LCD controller and LED backlight controller
lcd->setCursor(0,10); //bring cursor to top left corner
lcd->write("Temp:");
lcd->setCursor(0,0);
lcd->write("Light:");//print text
while(1)
{
Blinky
Button
ButtonInterrupt
Buzzer
LCD [selected] [IoT]
Binaries
Includes
src
LCD.cpp
Debug
LCD.launch
Problems
Console
IoT Sensor Support
Properties
Debug
LCD [C/C++ Remote Application] /home/iot/workspace_iot/LCD/Debug/LCD (06/11/16 10:49 am)
root@galileo:~# chmod 755 /tmp/LCD;/tmp/LCD ;exit
Target: ss
Target Name: 192.168.
Libraries status: Error

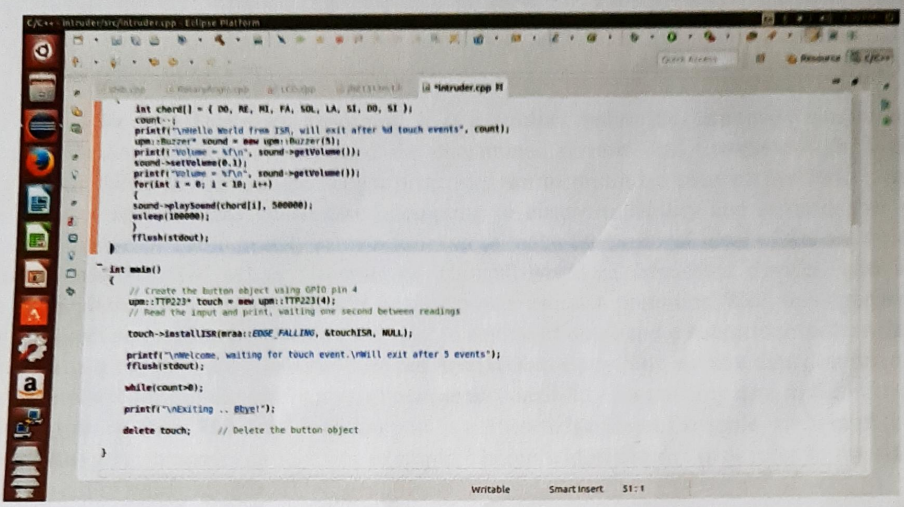






- We have integrated other two sensors namely the Buzzer and Touch Sensor with the project name Intruder. In this project when a touch is detected by the touch sensor hence a buzzer begins to buzz implying the user of an intrusion. The code of this project is as shown in the below images.





7. Connecting to Cloud using MQTT protocol

- **Introduction to MQTT**

MQTT stands for MQ Telemetry Transport. It is a publish /subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium. With this approach there is no direct connection between a publisher (a source of data) and a subscriber (access data requestor). Thanks to that, the publisher does not need to know anything about a data destination. It also does not need to handle any requests because responsibility for pushing data to the outside world is only on its side. The only common part is a knowledge about the topic, an identifiable name of a particular messages group. For example, "home/widows/state" may refer to the state (opened, closed) of windows in an intelligent house.

Reference :http://mqtt.org/

- **Publish / Subscribe**

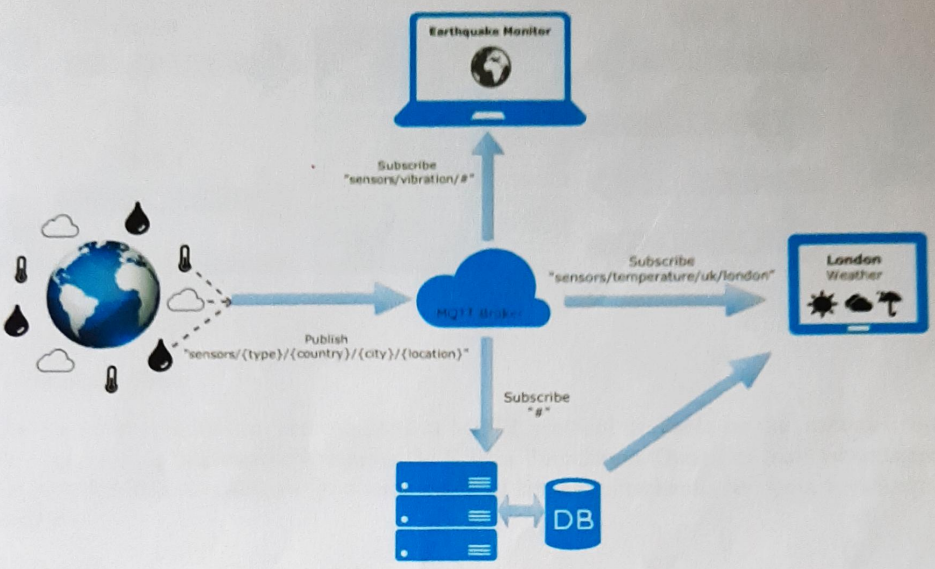
The publish / subscribe (often called pub-sub) pattern lies at the heart of MQTT. It's based around a message broker, with other nodes arranged around the broker in a star topology. This is a very different model to the standard client/server approach, and at first it might seem a little strange, but the decoupling it provides is a huge advantage in many situations.

Clients can publish or subscribe to particular topics which are somewhat like message subjects. They are used by the broker to decide who will receive a message. Topics in MQTT have a particular syntax. They are arranged in a hierarchy using the slash character (/) as a separator, much like the path in a URL. So a temperature sensor in your kitchen might publish to a topic like 'sensors/temperature/home/kitchen'.

Let's look at an example: Imagine a weather service which has a network of internet connected temperature sensors all over the world. All of these sensors maintain a connection to a broker and every ten minutes, they report the current temperature. They publish to a particular topic based on their location in the following format:

sensors/temperature/{country}/{city}/{street name}

So a sensor on Baker Street in London would publish to 'sensors/temperature/uk/london/baker_street' with a message containing the current temperature. The weather service wants to keep an up to date database of historical temperatures so it creates a database service which can subscribe to an MQTT topic. The database service will then be notified when a new temperature is received.



- **QoS**

MQTT is designed to work well on unreliable networks and as such provides three levels of Quality of Service for different situations. These allow clients to specify the reliability they desire.

QoS Level 0:

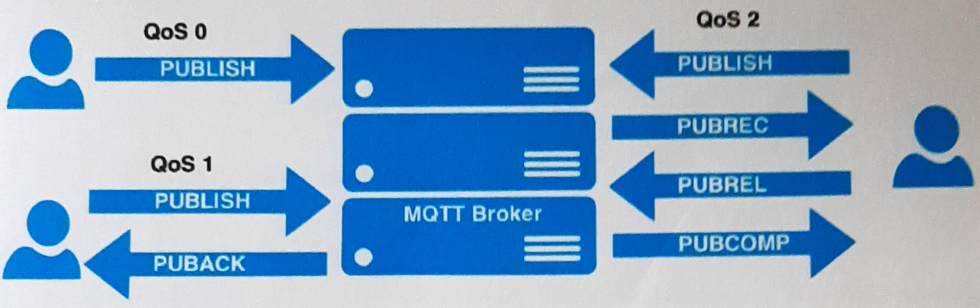
The simplest level of QoS, it requires no acknowledgment from the client and the reliability will be the same as that of the underlying network layer, TCP/IP.

QoS Level 1:

This ensures that the message is delivered to the client at least once, but it could be delivered more than once. It relies on the client sending an ACK packet when it receives a packet. This is commonly used for guaranteed delivery however developers must make sure that their systems can handle duplicate packets.

QoS Level 2:

This is the least common QoS, and ensures that a message is delivered once and only once. This method requires an exchange of four packets, and will decrease performance of the broker. This level is also often left out of MQTT implementations due to its relative complexity. Make sure you check this before choosing a library or broker.



- **Eclipse Paho**

There is a set of different implementations of MQTT protocol available on the market. Eclipse Paho – open source implementation under the Eclipse Foundation. One of its main advantages is that it provides implementations for a wide range of languages/technologies, from JavaScript to embedded C.

Reference: http://www.eclipse.org/paho/

The general workflow is described below.

a) Create an MQTT client instance.
b) Use MQTT client to connect with a specified broker.
c) Subscribe to a custom topic.
d) Publish a message from any sensor.

- **Conclusion**

MQTT is a fantastic protocol and its applications to IoT and M2M communications are endless. If you're looking for a very lightweight messaging system, it's a great choice and is likely to become very popular over the next few years.

References:

https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot

https://www.genuitec.com/introduction-to-mqtt-protocol/

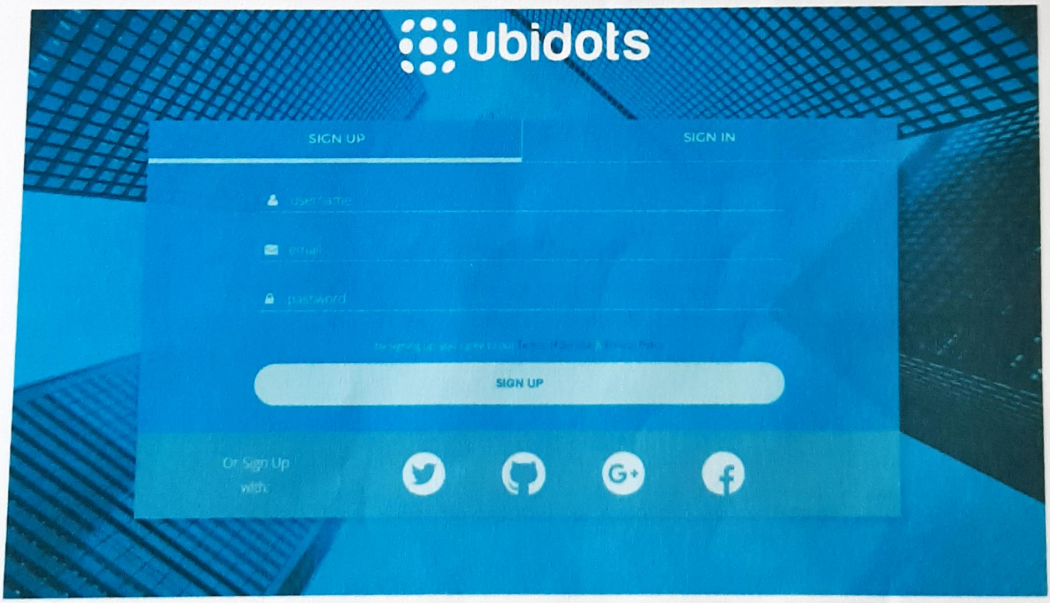
8. Dashboard configurations

Connecting your Galileo to Ubidots

For the cloud related exercises, we are using Ubidots as the cloud dashboard. We will be sending data from sensors to the Ubidots managed cloud and we will be able to see real time data on their dashboard. Ubidots also has the capability to add various widgets on their dashboard for improved visualization of data. You can also set rules to achieve actuations like HTTP post, email, SMS and change value of other variables based on rules. This will help to achieve an end to end implementation of IoT exercises.

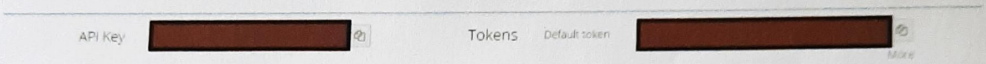
1. Log on to http://ubidots.com/

2. Sign up and create an account

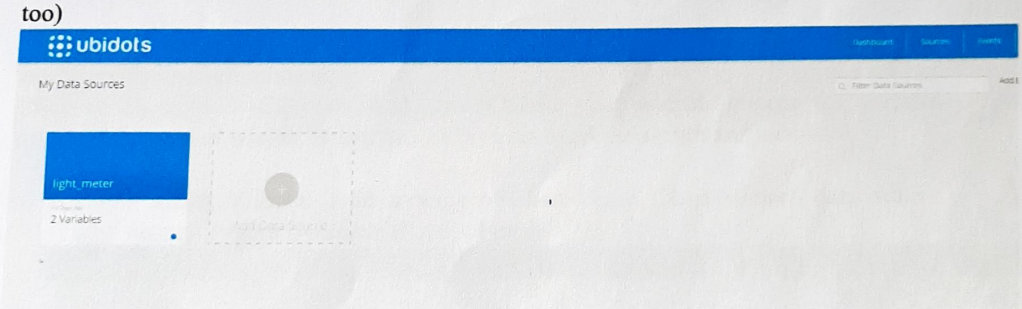


3. Once you have logged in, please go to "My Account" and click on "API Credentials"

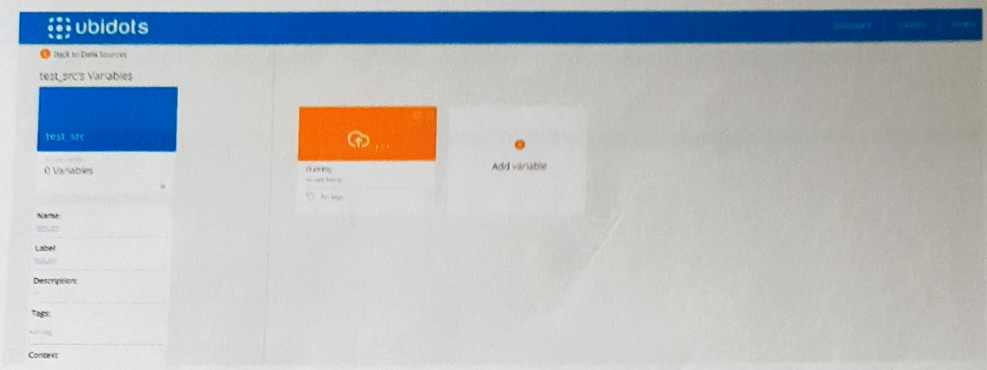
4. Note down your Default Token and your API Key. These will be used to connect to the dashboard from Eclipse. Refer to section "Code Snippet" at the end of this document.



5. Creating a data source and a variable to publish data.
   Click on Sources->Add Data Source->Name the data source (Light_meter). The data source name should match with the source specified in code (spelling and case sensitive too)

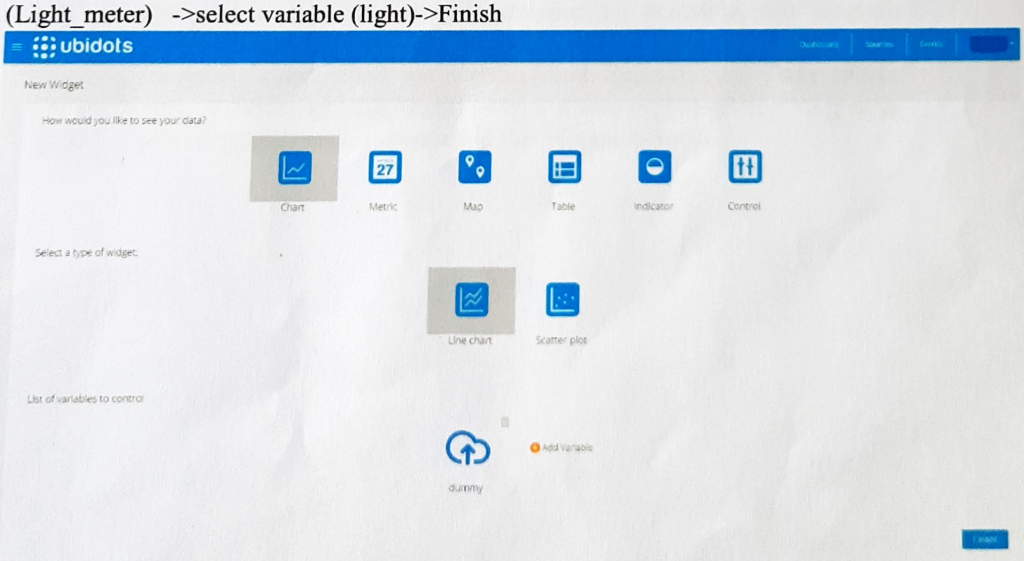


6. Add a variable in the data source
   Click on the data source->Add a variable->Name the variable (light)->choose the type of variable (default). The variable name should match with the source specified in code (spelling and case sensitive too). Similarly create a variable named Buzzer which will be used in the example.

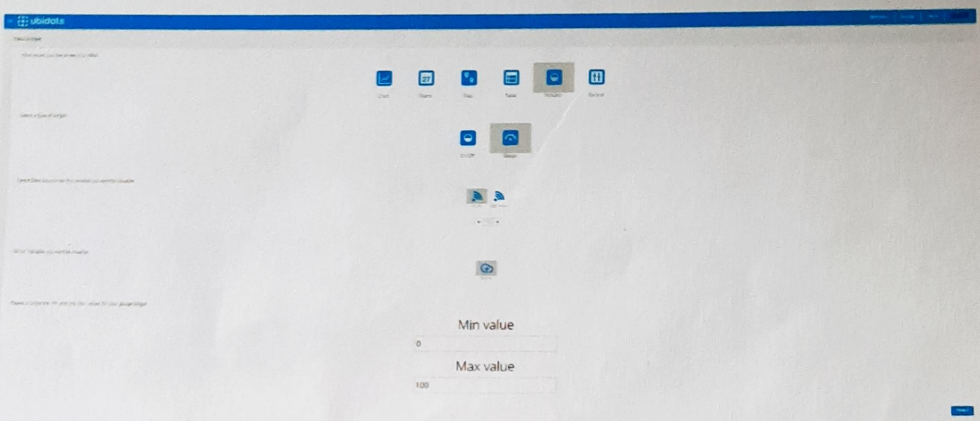


7. Adding widgets to Dashboard
   Click on Dashboard->Click on Add sign->Choose your widget->choose the variable based on the type of widget. Below are 2 examples to create a chart and a control.

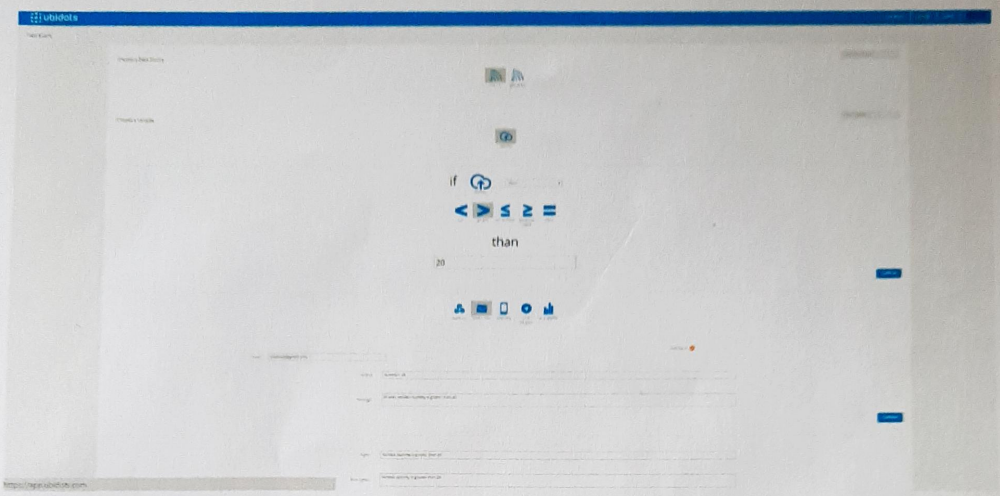
   a. Creating Chart Widget: Add a widget->Chart->Line Chart->Select data source (Light_meter) ->select variable (light)->Finish



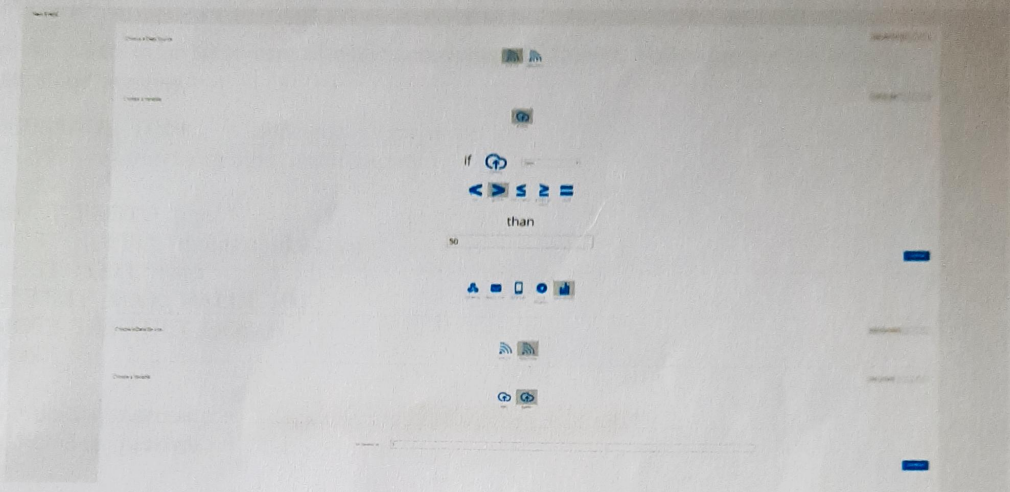
b. Creating an Indicator Widget: Add a widget->Indicator->type of indicator (Gauge) ->data source (Light_meter)->variable (light)->specify min (0) and max value (100)->finish



8. Adding an Event to the Dashboard: Event like sending an email, SMS, HTTP Post or actuation can be created using simple rules in the dashboard. Below are 2 examples of creating a rule for sending an email and enabling actuation. For actuation, your program must subscribe to a variable and implement callbacks.
   a. Creating an email event: Click on Events->Add Event->Select Data source (Light_meter) ->Select variable (light)->Make a rule (< 10)->Select trigger as Email->Enter recipient's email, subject and Description->Finish



b. Setting a value to a variable: Click on Events->Add Event->Select Data source (Light_meter)->Select variable (light)->Make a rule (< 10)->Select trigger as "Set a variable"->Select Data source (Light_meter)->Select Variable (Buzzer)->Set value (1) ->Continue->Finish



Code Snippet

These are the points to be taken care of while connecting to Ubidots. Following are the settings done in the cloud exercise:

```
#define MESSAGE_TOPIC "/v1.6/devices/Light_meter"
#define TOPIC "/v1.6/devices/light_meter/Buzzer/lv"

#define HOST_PROTO "tcp://"
#define HOST_SUFFIX "things.ubidots.com:"
#define HOST_PORT "1883"
#define DUMMY_MAX_VALUE 100
#define MQTT_DEFAULT_QOS 0
#define QOS 1

constchar * ubidots_username = "Default Token as described in point 4";
constchar * ubidots_password = "";
```

MESSAGE_TOPIC – This is the name of the data source that has been created. In the exercise example, name of data source is "light_meter". The name mentioned in code and in Ubidots should match even in case.

TOPIC – In the exercise example, we have created a variable "buzzer" under data source "light_meter" whose value will be changed to 1 when light value is less to achieve actuation as shown in example 8a. The name mentioned in code and in Ubidots should match even in case.

HOST_PROTO, HOST_SUFFIX & HOST_PORT – These are settings required to connect to Ubidots. Please keep the settings unchanged.

MQTT_DEFAULT_QOS & QOS – These are QOS values set for publishing and receiving messages. Please refer to PAHO MQTT APIs for more details.

ubidots_username – This is the default token as described in section 4 of this document. This is used as the credential to connect to Ubidots. This is match exactly as shown in Ubidots.

The web example needs the following to be created in Ubidots dashboard:

1. Data Source named Light_meter
2. Variables named light and Buzzer inside the data source Light_meter
3. Widgets like Charts and Gauge for seeing live light data
4. Events for triggering SMS when light < 10 and changing Buzzer value to 1 when light < 10 and changing Buzzer value to 0 when light >= 10

MQTT API Sequence followed in exercise Example

The general workflow is described below.

- Create an MQTT client instance.
- Use MQTT client to connect with a specified broker.
- Subscribe to a custom topic.
- Publish a message from any sensor.

For MQTT API reference, please visit: http://ubidots.com/docs/api/#mqtt-api-reference

a. **MQTTClient_create-** This function creates an MQTT client ready for connection to the specified server and using the specified persistent storage. Param handle: A pointer to an MQTTClient handle. The handle is populated with a valid client reference following a successful return from this function.

b. **MQTTClient_setCallbacks-** This function sets the callback functions for a specific client. If your client application doesn't use a particular callback, set the relevant parameter to NULL. Calling MQTTClient_setCallbacks() puts the client into multi-threaded mode. Any necessary message acknowledgements and status communications are handled in the background without any intervention from the client application.

c. **MQTTClient_connect-** This function attempts to connect a previously-created client to an MQTT server using the specified options. If you want to enable asynchronous message and status notifications, you must call MQTTClient_setCallbacks() prior to MQTTClient_connect(). Param - handle A valid client handle from a successful call to MQTTClient_create().

d. **MQTTClient_subscribe-** This function attempts to subscribe a client to a single topic. This call also specifies the QOS requested for the subscription.

e. **MQTTClient_publish-** This function attempts to publish a message to a given topic. An MQTTClient_deliveryToken is issued when this function returns successfully. If the client application needs to test for succesful delivery of QoS1 and QoS2 messages, this can be done either asynchronously or synchronously.

f. **MQTTClient_disconnect-** This function attempts to disconnect the client from the MQTT server. In order to allow the client time to complete handling of messages that are in-flight when this function is called, a timeout period is specified. When the timeout period has expired, the client disconnects even if there are still outstanding message acknowledgements. The next time the client connects to the same server, any QoS 1 or 2 messages which have not completed will be retried depending on the clean session settings for both the previous and the new connection.

MQTTClient_destroy - This function frees the memory allocated to an MQTT client.

PROCOM ENTERPRISES

9/302/16, 25B/32B,Adinath Co. Operative Society ,Near Sahakar Nagar
ICHALKARANJI - 416115
Cell No. 09422628628 /09342930555
Email – procom.enterprises@yahoo.in

Steps for connecting gateway to a wifi

1. Open the terminal. [Ctrl+Alt+T] or click on first icon & type terminal and open the terminal icon.
2. Write statement on screen=> sudo screen /dev/ttyUSB0 115200
3. Enter password=> iot123
4. Plug in power supply.
5. Wait for Galileo login instruction.(press enter)
6. galileo login: root (press enter).
7. root@galileo# connmanctl (press enter).
8. connmanctl> enable wifi(press enter).
9. connmanctl> scan wifi(press enter) .
10. connmanctl> services.
11. System gives list of SSID codes copy that code (select SSID of your choosed wifi).
12. connmanctl>config (give space & paste the SSID code). (press enter)
13. connmanctl>agent on(press enter)
14. connmanct>connect(give space & paste the SSID code).(press enter)
15. Passphrase? (type password of wifi)
16. connmanctl> exit
17. root@galileo#ifconfig (getting IP adderess)
18. copy IP address or write down that IP address on paper.
19. root@galileo#exit

NAME :

MOBILE NO:

1.

2.

3.

4.

SR NO	PARTICULARS	QUANTITY	Check In	C C
1	INTEL GALILEO GEN 2	1	✓	
2	32GB SOFTWARE PENDRIVE	1	✓	
3	ELEVEN SENSOR& ACTUATORS BOX	1	✓	
4	NANO ADAPTER	1	✓	
5	ANTENNA	2	✓	
6	ADAPTER	1	✓	
7	ETHERNET CABLE	1	✓	
8	USB TO FTDI CABLE	1	✓	
9	8GB MICRO SD CARD	1	✓	
10	Getting Started Guide eIoT3500	1	✓	

SIGNATURE OF MATERIAL RECIEVER

FOR PROCOM ENTERPRISES