

IMPLEMENTATION OF SMART GPS FOR TRACKING OF BOAT AND HEALTH MONITORING

Submitted in partial fulfilment of the criteria for a Bachelor of Engineering degree in
Electronics and Communication Engineering
by

SHAIK MOHAMMED FERAZ (Register.No.37130370)
RALLAMADUGU VINAY KRISHNA (Register.No.37130338)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING SCHOOL OF ELECTRICAL AND ELECTRONICS**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

APRIL 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **SHAIK MOHAMMED FERUZ (Reg.No.37130370)** and **RALLAMADUGU VINAY KRISHNA (Reg.No.37130338)** who carried out the project entitled **"IMPLEMENTATION OF SMART GPS FOR TRACKING FOR BOAT AND HEALTH MONITORING"** under our supervision from September 2020 to April 2021.

Internal Guide

Dr. R. M. Joany, M.E., Ph.D.,

Head of the Department

Dr. T. RAVI M.E., Ph.D.

Submitted for Viva voce Examination held on 19/04/2021

Internal Examiner

External Examiner



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DECLARATION

We, **SHAIK MOHAMMED FEROZ (Reg.No.37130370)** and **RALLAMADUGU VINAY KRISHNA (Reg.No.37130338)** hereby declare that the Project Report entitled **"IMPLEMENTATION OF SMART GPS FOR TRACKING OF BOAT AND HEALTH MONITORING"** done by us under the guidance of Dr. R. M. Joany, Ph.D., is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

1. SHAIK MOHAMMED FEROZ
2. RALLAMADUGU VINAY KRISHNA

DATE: 16-04-2021

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. N. M. NANDHITHA M.E., Ph.D.** Dean, School of Electronics and Communication Engineering and **Dr. T. RAVI M.E., Ph.D.** Head of the Department, Department of Electronics and Communication Engineering for providing me necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Dr. R. M. Joany, Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the **Department of Electronics and Communication Engineering** who were helpful in many ways for the completion of the project.

We would like to thank our beloved **PARENTS** for their constant love and support, without whom this wouldn't be and finally we would like to thank the **ALMIGHTY** for his wonderful blessing.

ABSTRACT

Fishermen are crossing borders and identifying places in the sea is becoming a difficult task with the current latest equipment to them, as a result of which they cross borders. We hear of many Tamil fishermen being apprehended and detained by Sri Lankan naval forces on a daily basis; the key explanation for this offence is that the sea boundary between the two countries is not easily recognisable.

Furthermore, in the event of an impending natural disaster, failure or delay in notifying affected workers to evacuate results in a large-scale loss of life. In this paper, we suggest a method for protecting fishermen by logging their arrivals and departures from the harbour.

Using an integrated system and the Global Positioning System (GPS), Geo Fencing, and Mobile Systems, they are notified of the country's sea boundary.

We use GPS to keep track of the fishermen's current position. The GPS's current latitude and longitude coordinates are sent to the database, which the administrator uses for continuous tracking and monitoring of the user. If the user is in danger, current position predictions can be made using their credentials and last known location.

Another advantage is that the logging procedures assist official authority agencies in identifying fishermen and their activities for the protection and security of the fishermen.

TABLE OF CONTENTS

ABSTRACT	V
1. INTRODUCTION	1
2. LITERATURE SURVEY	1
3. Existing system	4
Proposed system	4
Block diagram	5
Specification:	6
Using Arduino IDE	7
HEARTBEAT SENSOR	13
Overview	30
UBIDOTS EXPLORER	40
Introduction	40
How Ubidots Works?	40
Values	41
Timestamps	41
Context	42
Time Series	42
HTTP	43
API URLs	45
HTTP	46
HTTPs	46
Fixed IPs	46
HTTP Authentication	47
Send data	48
Retrieve data	53
HTTP Examples	57
Sending a geopoint	57
Sending a string value	57
Sending values in bulk from a datalogger	58
Retrieve the last value for control	58
Response codes	59

MQTT	60
Broker URLs	61
Non-TLS	61
TLS	61
MQTT Authentication	62
Send data (Publish)	63
MQTT Examples	66
GPS Tracker	67
Weather description	67
Retrieve the last value for control	67
TCP / UDP	68
TCP Authentication	68
TCP URLs	69
Non-TLS	69
TLS	69
Send Data	70
4. RESULT	75
5. CONCLUSION	78
REFERENCES	79

List of Figures

Figure No	Figure	page no
1.	Block diagram Transmitter	5
2.	Block Diagram Receiver	5
3.	ESP8266 Nodemcu DevKit	6
4.	Pin Configuration of ESP8266	7
5.	Using Aurdino IDE	8 - 11
6.	Blink Test	11
7.	PIN Assignment of Blink Test	12
8.	Working of Heart Beat Sensor	15
9.	Flashing Nodemcu Firmware	16
10.	Thermocouple	19
11.	Application creating sensor from Scratch	21
12.	Temperature Sensor	21
13.	Pressure Sensor	22
14.	Layered architecture of IOT	24
15.	Functional components of a middleware for IOT	26
16.	Ubidot Explorer working	54
17.	Output Transmitter	67
18.	Output Receiver	67
19.	Notification Alert	69
20.	Atmospheric Measurements	69

INTRODUCTION

A human's life has always required the use of security systems and navigators. Advances in modern electronics have ushered in radical improvements in these areas. Theft of equipment is a serious problem in many industries, including transportation and construction, especially for larger companies. There is no way to track the vehicle's movement, and we have no idea what the actual state and location of the vehicles are. One of the most obvious ways to keep track of vehicles is to use a GPS-based vehicle tracking device. The Global Positioning System (GPS) is a satellite-based navigation system that provides accurate location and time information in every weather situation on the planet at all times. We want to solve the problem of vehicle tracking by using a GPS-based monitoring device with geo-fencing capabilities. This system consisted of a few major components that worked together just to communicate the vehicle's position to a remote user through satellite communication. Remote control of geographic areas enclosed by a virtual fence (geo-fence) and automatic detections as mobile objects entered or exited these areas is made possible by geo-fencing. Geo-fence software and tools kept track of which devices or other physical objects entered or left the geo-fenced area and granted administrators access to information.

LITERATURE SURVEY:

1. GPS Based Location Monitoring System with Geo-fencing Capabilities

Author: A.H. Abbas^{1, a)}, Mohammed I. Habelalmateen^{1, 2, b)}, Syukran Jurdi^{1, c)}, L. Audah^{1, d)} and N.A.M. Alduais

A research on a GPS-based location monitoring system with geo-fencing capabilities is presented in this paper. This system included a high-security system that guarded against vehicle theft. It also used the Internet of Things to send a notification to the user based on the location's boundary (IoT). The device in this study was able to easily monitor and track the position of the vehicle, as well as issue a warning when it left the geofence area. The hardware and software components of this device is segregated. The ESP8266 NodeMCU and GPS module were the hardware components, while Google Maps and the IoT platform were the software components. The admin could

track the vehicle via computer, and when the vehicle left or entered the geofence area, a warning alert was sent to the admin's registered email address. The prototype system was put to the test by driving around the geofence area. When the vehicle left or approached the boundaries, the results indicated the correct location of the vehicle and an email notification warning. When compared to the real-map on a cell phone, the location accuracy is about 95%.

2. Design Of Android Based On Children Tracking Location Monitoring System

Author: Marti Widya Sari, Desinta Ningrum Belsa Putri, Banu Santoso

Any parent has a number of duties or responsibilities that they cannot stop. This practise limits parental control and oversight of their children's activities. One of the most common issues is parents' fears over tracking the appearance and monitoring of their children's behaviours. Children must first cultivate a sense of security in order to be prepared to confront this often threatening environment. It's normal to remind a child of the risk of being kidnapped or defrauded by someone he has never met or is already acquainted with on occasion, without overprotecting him. Improved child supervision is required without limiting the child's rights. Using a Geo Positioning System as one option for monitoring the presence, location, and accuracy is very high (GPS). This GPS is bundled in a flexible format to make it easy for parents to supervise their children in the places that are already available to them and to locate the child's whereabouts if they become separated from their parents. The Google API optimised technology is extremely useful in monitoring because it allows parents to predict the tracker's approximate location. The device will transmit the geographic details of children who have gone missing or been separated from their parents to mobile apps. This GPS belt technology has proven to be useful in assisting parents in monitoring or locating their children.

TRACKING AND WARNING SYSTEM FOR FISHERMEN USING IOT

Author: R.VinothKumar¹, M.SuriyaPrakash², P.Andril³, M.ThamizhIni⁴, N.VivekKumar⁵

The issues of fishermen stray in each territorial water has come as an potential irritant in the bilateral relations between the neighboring states and countries. International

Marine Time Boundary line (IMTL) between countries will always have security problem. Due to carelessness the fishermen may accidentally cross the country border. In such situation they may face attack from the opposite navy. To avoid such situation a device using embedded system has been designed to protect the fishermen. The fishermen's current location is determined using a GPS receiver. The heart beat sensor continuously monitors Heartbeat of the fishermen. GSM is used to transfer the data to the cloud storage using IOT. If fishermen navigate beyond the country border a warning message is send from the control room. This device helps the fishermen to handle any hazardous situation and to improve the safety.

3. Location-based services using Geofencing

Author: Abhishek Singh¹ , Ankit Pal² , Divyansh Garg³ , Dolly Yadav⁴

Geo-fencing (geofencing) is a useful function in applications that defines territorial borders using the Global Positioning System (GPS) or radio frequency identification (RFID). Geofence is, in reality, a virtual firewall. Geofencing is a cutting-edge technology that enables people to conveniently discover fascinating services, connect to them, and enable vendors to sell their services for a range of purposes, including electronic toll collection, contextual advertisement, and visitor information systems, often without the need for external infrastructure. The primary goal of this study was to determine how the use of spatial data would increase consumer advertisement efficiency. Tracking and tracking systems focused on satellite-based global navigation services with geofencing capabilities may help pinpoint an institution's or company's exact location and boost revenue and market opportunities. They will also advertise on tablets instead of big billboards, which is more cost-effective and reliable.

4. DEVELOPMENT OF AUTOMATIC GEOFENCING AND ACCIDENTAL MONITORING SYSTEM BASED ON GPS TECHNOLOGY

Author: Danish karim¹ and Jaspal Singh²

The key goal of this project is to build and build an integrated device for a vehicle that can detect a road crash and provide emergency medical assistance as well as provide protection for robbery prevention by using a single shock sensor. In the event of a collision, the device could instantly send a text message to the emergency response team, signalling the location of the ambulance. When there is no one in the car, the

vehicle can go into protection mode automatically. This device could give a warning message to the owner if someone wants to rob the car. The framework uses AT89S52 as a central hub for geofencing and unintended tracking. This includes a shock counter, liquid crystal display (LCD), dual relay, and wireless networking, among other innovations. Geo fencing equipment is also used in this framework. Geo fencing allows for the remote control of predefined borders and the automated detection of vehicle departure.

Existing system:

A geofencing and accidental detection scheme built on AT89S52 as a central core to track traffic accidents and provide emergency medical assistance, as well as for surveillance and theft prevention. In the event of a collision, the device could instantly send a text message to the emergency response unit, showing the vehicle's location. When there is no driver, the vehicle enters security mode, which uses a shock sensor to provide security.

Drawbacks:

The existing system does not have any security system to avoid thefts which is being added in the proposed system.

Proposed system:

- The system aims at monitoring the fisherman current location, when the fisherman enters the Geo fence. And he leaving the fencing then automatically send an alert SMS / Mail to the parents as well as represents.
- Geo fences are basically virtual regions that are considered around geo point with a specific radius.
- And also we can monitor the health condition of the fisherman like temp sensor is used to measure the body temperature, heartbeat sensor is used to measure the pulse rate of the fisherman and also pressure sensor is used to measure the pressure level of the sea. So we can see the fisherman condition and take immediate actions.

Block diagram:

Transmitter section

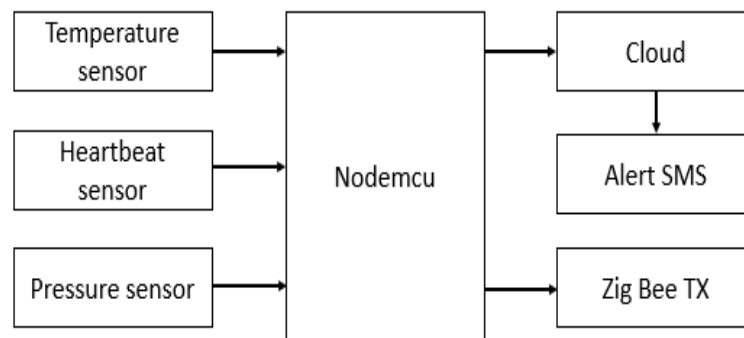


Fig no 1

Receiver section

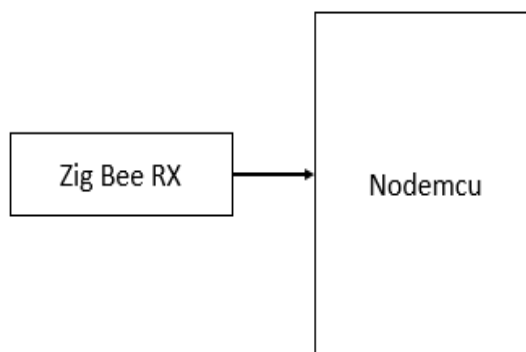


Fig no 2

Required components:

Hardware requirements:

Nodemcu

Temperature sensor

Heartbeat sensor

Pressure sensor

Zig Bee pair (TX & RX)

Software required:

Arduino ide

Embedded C

HARDWARE REQUIREMENTS:

HARDWARE SETUP DETAILS:

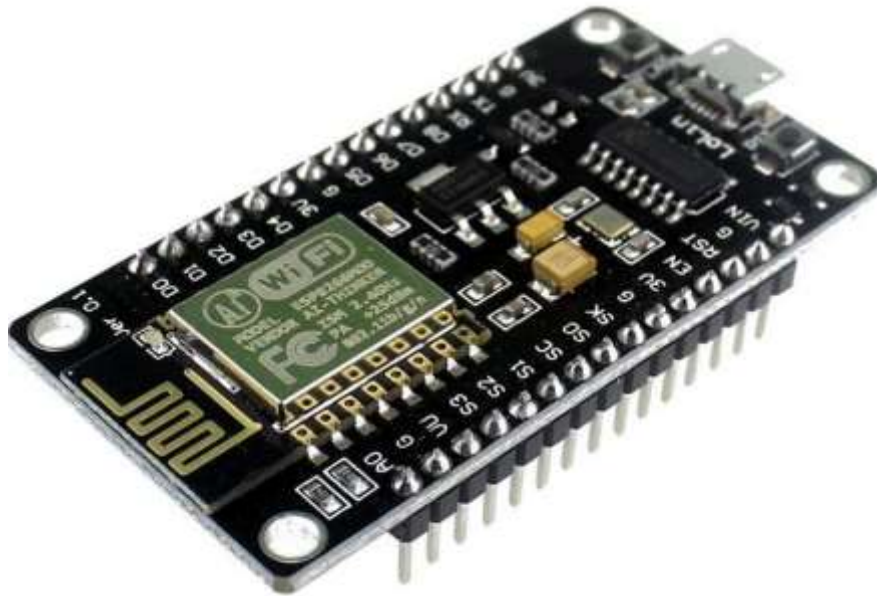


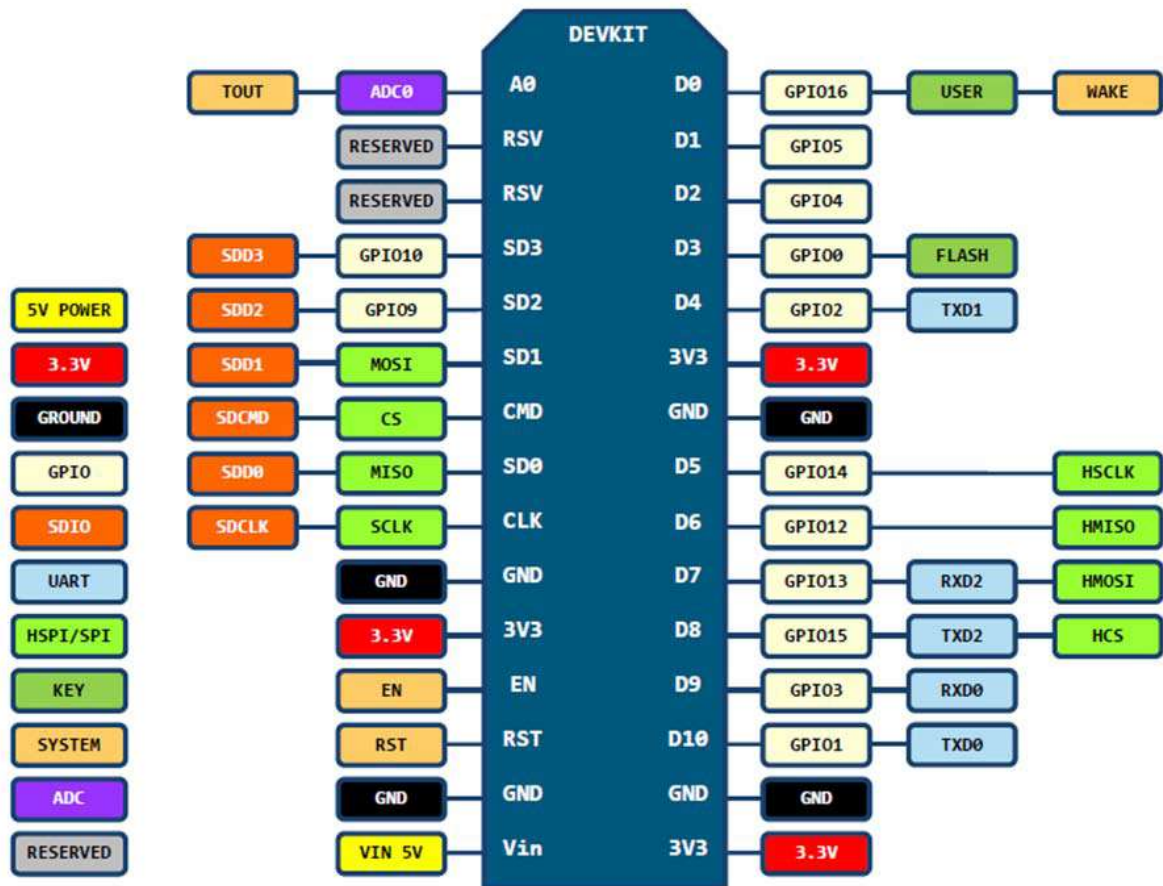
Fig 3:ESP8266 Nodemcu Wi-Fi Devkit

Specification:

- The voltage is 3.3 volts.
- Soft-AP with Wi-Fi Direct (P2P).
- Current intake ranges from 10uA to 170mA.
- Attachable flash memory: 16MB maximum (512K normal).
- TCP/IP protocol stack is integrated.
- Tensilica L106 32-bit processor
- Processor frequency: 80–160 MHz
- RAM: 32 kilobytes plus 80 kilobytes
- 17 GPIOs (multiplexed with other functions).
- Analog to Digital: 1 input with a resolution of 1024 steps. • In 802.11b mode, performance capacity is +19.5dBm.
- 802.11 b/g/n support

Maximum concurrent TCP connections: 5.

Pin Definition:



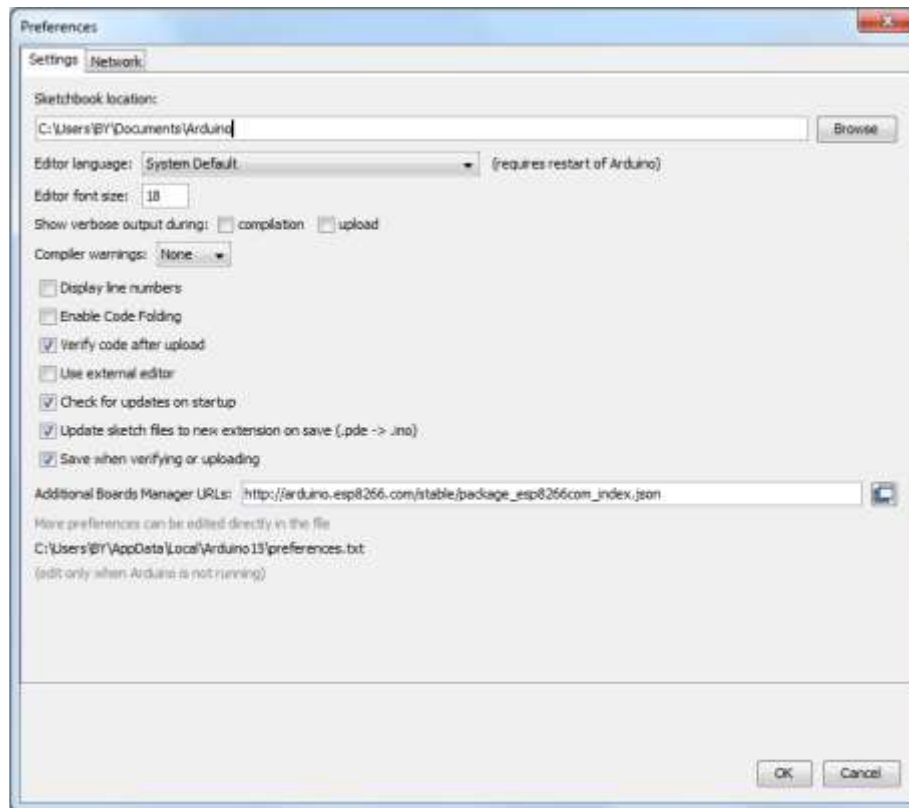
D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Fig no:4 Pin Configuration

Using Arduino IDE

Since the ESP8266 module is essentially a WiFi/Serial transceiver, the most basic way to use it is through serial commands. This, though, is inconvenient. The Arduino ESP8266 project, which is a tweaked version of the Arduino IDE that you must run on your motherboard, is what we suggest. Since we'll be using the well-known Arduino IDE, this allows using the ESP8266 device a breeze. Install the ESP8266 library in the Arduino IDE environment by following the steps below. Install the ESP8266 Board Package

In the Additional Board Manager URLs area of the Arduino v1.6.4+ preferences, type http://arduino.esp8266.com/stable/package_esp8266com_index.json.



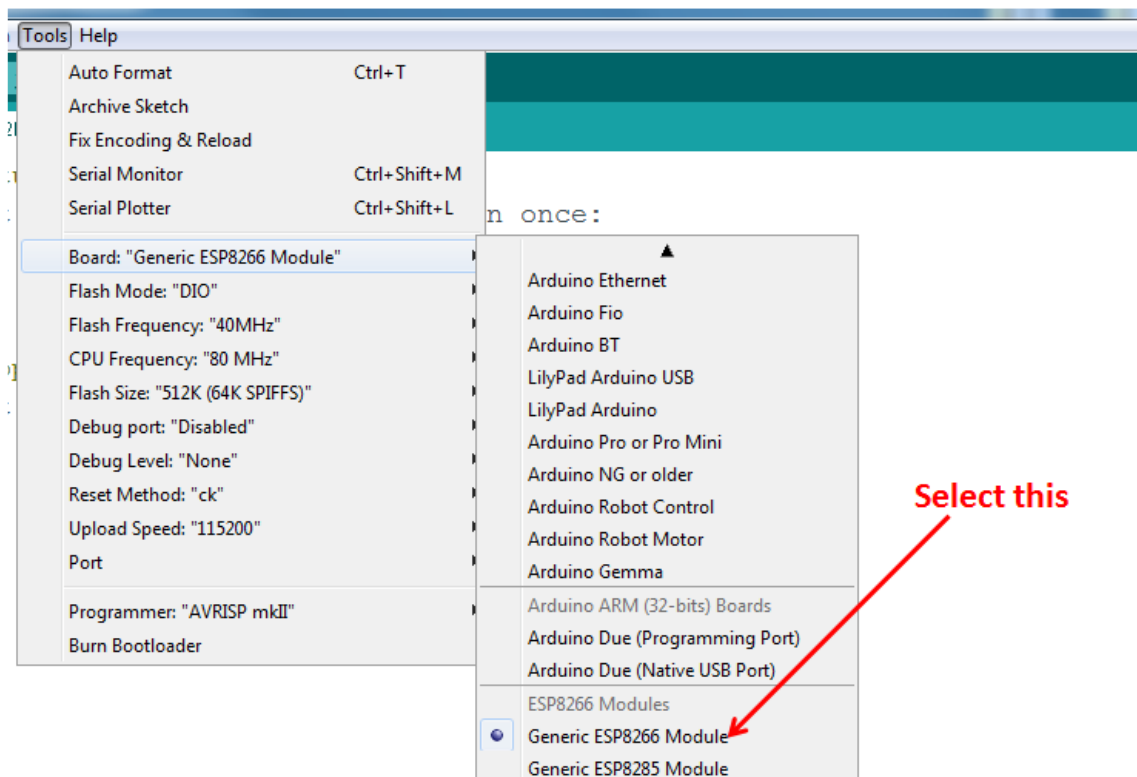
To get to this panel, go to 'File' -> 'Preferences.' Next, load the ESP8266 kit using the Board manager.



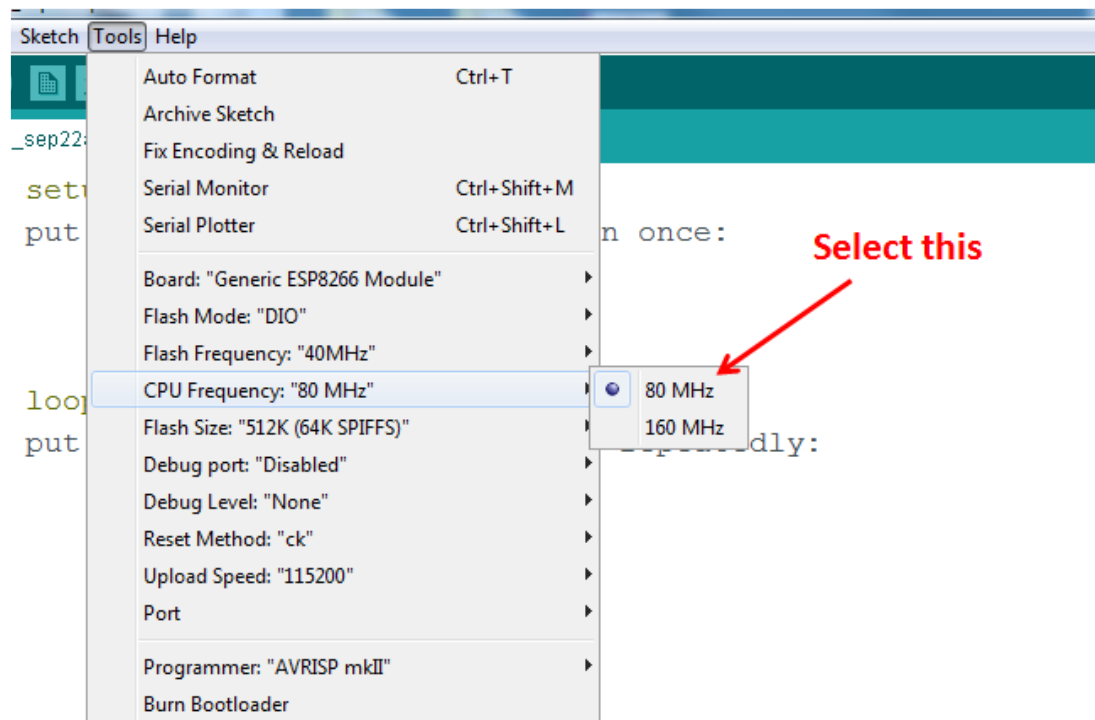
To enter this line, go to 'Tools' -> 'Board:' -> 'Board Manager...'. To instal the ESP8266 library kit, scroll down to ' esp8266 by ESP8266 Community ' and press the "Install" tab. Close and reopen Arduino IDE until the installation is complete for the ESP8266 library to take impact.

Setup ESP8266 Support

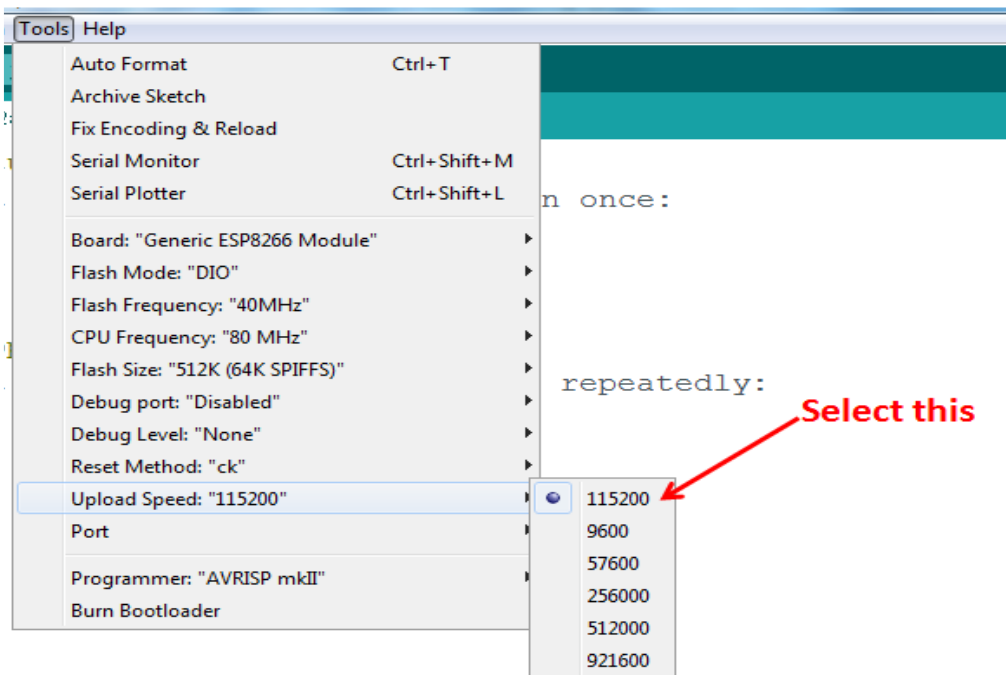
When you've restarted Arduino IDE, select 'Generic ESP8266 Module' from the 'Tools' -> 'Board:' dropdown menu.



Select 80 MHz as the CPU frequency (you can try 160 MHz overclock later)

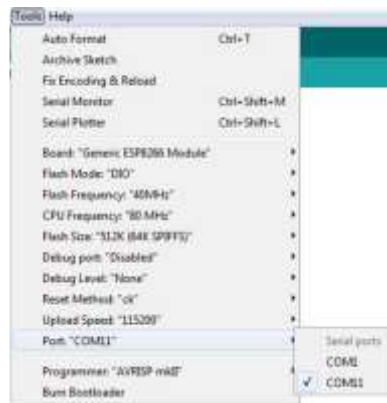


Select '115200' baud upload speed is a good place to start - later on you can try higher speeds but 115200 is a good safe place to start.



Go to your Windows 'Device Manager' to find out which Com Port 'USB-Serial CH340' is assigned to. Select the matching COM/serial port for your CH340 USB-Serial interface.





Blink Test

Let's start with a quick blink test. Fill in the blanks in the drawing window (and save because you'll need to). Figure 3-1 shows how to attach an LED.

The board must now be moved into bootload mode. This must be done before each upload. There is no time limit in bootload mode, so there is no need to hurry!

- Press and hold the 'Flash' button.
- Hit the 'RST' button while holding down the 'Flash' button.
- Press the 'RST' button, then the 'Flash' button.

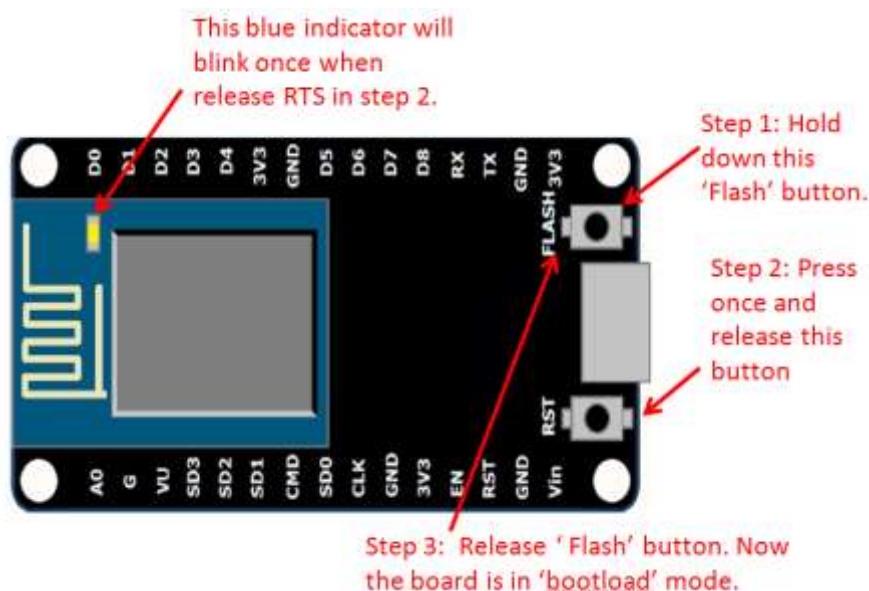
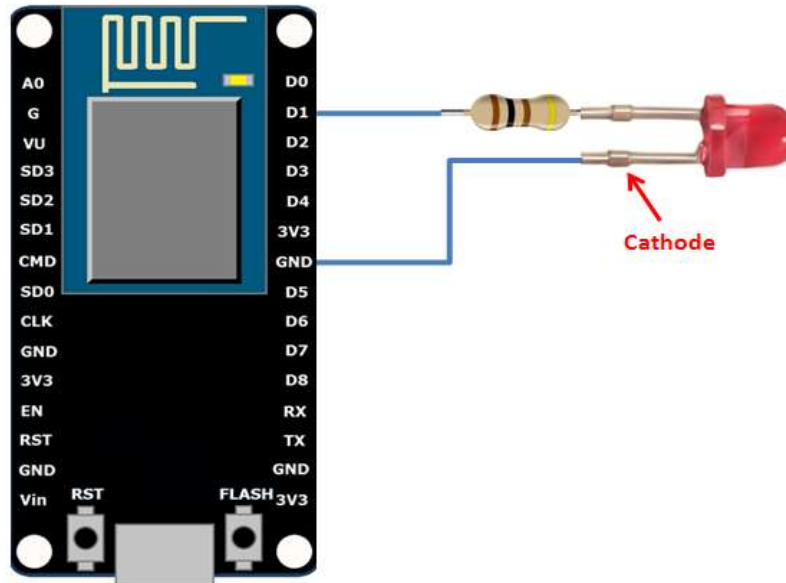


Fig No 4:blink test

Once the ESP board is in bootload mode, upload the sketch via the IDE



Connection diagram for the blinking test

```

void setup() {
  pinMode(5, OUTPUT); // GPIO05, Digital Pin D1
}

void loop() {
  digitalWrite(5, HIGH);
  delay(900);
  digitalWrite(5, LOW);
  delay(500);
}

```

Uploading...

WARNING: Spurious .github folder in 'Adafruit IO Arduino' library
 WARNING: Spurious .tests folder in 'Adafruit IO Arduino' library
 WARNING: Spurious .github folder in 'Adafruit MQTT Library' library
 WARNING: Spurious .github folder in 'Adafruit IO Arduino' library
 WARNING: Spurious .tests folder in 'Adafruit IO Arduino' library
 WARNING: Spurious .github folder in 'Adafruit MQTT Library' library

Sketch uses 223,197 bytes (51%) of program storage space. Maximum is 434,160 bytes.
 Global variables use 31,572 bytes (38%) of dynamic memory, leaving 50,348 bytes for local v
 Uploading 326392 bytes from C:\Users\BY\AppData\Local\Temp\buildb7f3357d9ec330fa2a4043584dd
 [36%]

Flashing NodeMCU Firmware on the ESP8266 using Windows

NodeMCU is a firmware that allows you to program the ESP8266 modules with LUA script. And you'll find it very similar to the way you program your Arduino. With just a few lines of code you can establish a WiFi connection, control the ESP8266 GPIOs, turning your ESP8266 into a web server and a lot more. In this

tutorial we are going to use another ESP8266 module with pin header adapter board which is breadboard friendly.



Fig no 5:ESP8266

ESP8266 Module Breadboard Friendly with Header Connector

Pin Assignment:

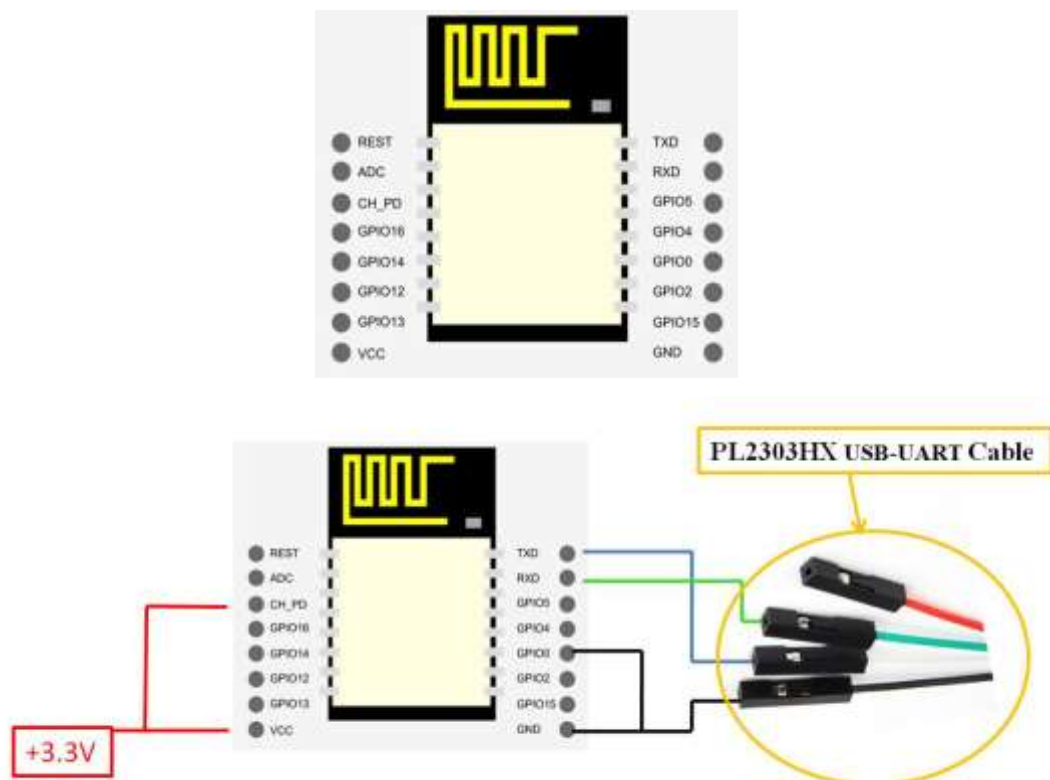


Fig no 6:ESP8266 Pin Assignment

HEARTBEAT SENSOR

The rhythm of a person's pulse is the contraction or expansion of the valves of his or her heart as they force blood from one area to another. The heart beat rate is

the amount of times the heart beats per minute (BPM), and the pulse is the heartbeat that can be detected through any artery close to the skin.

Two Ways to Measure a Heartbeat

Manual Way: Manually measuring one's rhythms at the wrist (radial pulse) and the neck (neck pulse) will reveal the presence of a heartbeat (carotid pulse). The technique entails placing the two index and middle fingers on the wrist (or neck below the windpipe) and counting the number of pulses for 30 seconds before multiplying the number by two to determine the heart rate. However, only light touch can be used, and fingertips should be rotated up and down before a heartbeat is detected.

- Using a sensor: The optical strength difference as light is scattered or absorbed during its journey through the blood as the heart rhythm varies may be used to calculate heart rate

The Heartbeat Sensor's Principle

- The photo plethysmography concept is used to power the heartbeat sensor. It tracks the difference of blood flow through every organ of the body that results in a shift in the strength of light passing through that organ (a vascular region). The pacing of the pulses is more critical in systems where the heart rhythm is being tracked. The amplitude of heart pulses determines the distribution of blood volume, and since light is absorbed by blood, signal pulses are equal to heart beat pulses.

There are two types of photoplethysmography:

Light from the light emitting unit is transferred through every vascular area of the body, such as the earlobe, and is detected by the detector.

Reflection: The regions reflect the light produced by the light emitting system.

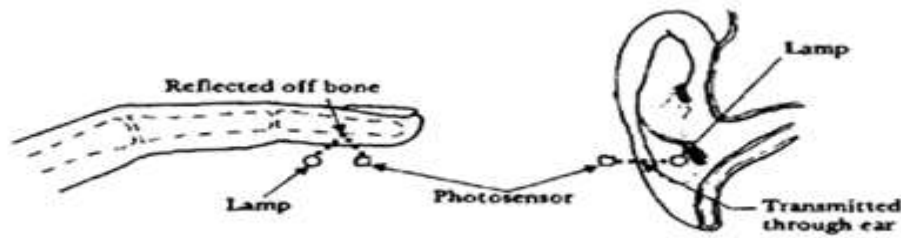
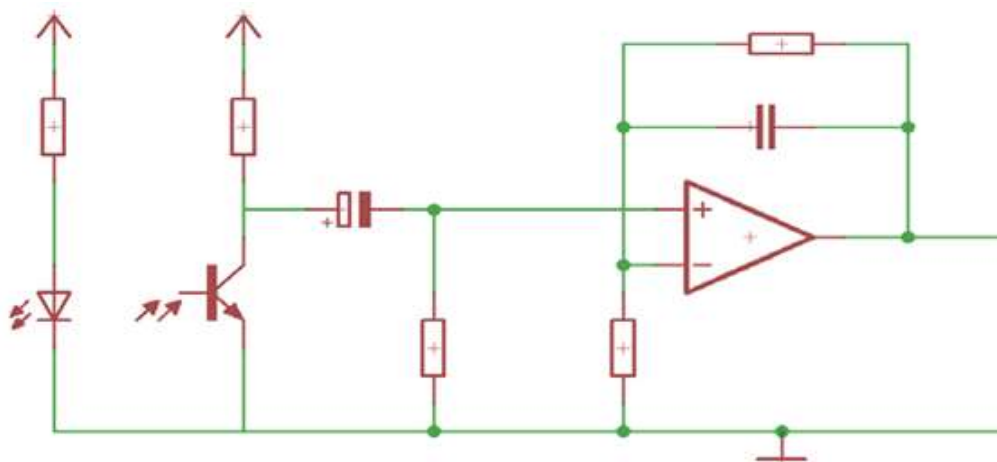


Fig no 7 :Working of a Heartbeat Sensor

A light emitting diode and a detector, such as a light sensing resistor or a photodiode, make up a simple heartbeat sensor. The flow of blood to various parts of the body varies as the heart rhythm pulses. When a tissue is lit by a light source, such as the light released by a led, it either absorbs (a finger tissue) or transmits the light (earlobe). The blood absorbs some of the light, and the light detector receives the emitted or reflecting light. The amount of light absorbed is determined by the tissue's blood volume. The contribution of the detector is an electrical pulse that is equal to the heart rhythm.

This signal is essentially a DC signal relating to tissues and blood flow, with an AC portion superimposed over it that is synchronous with the heart beat and triggered by pulsatile changes in arterial blood volume. As a result, the most important prerequisite is to isolate the AC portion.



To The output of the detector is first filtered using a two-stage HP-LP circuit before being converted to digital pulses using a comparator circuit or a basic ADC to obtain the AC signal. The digital pulses are sent to a microcontroller, which uses the formula $BPM(Beats\ per\ minute) = 60 \cdot f$ to calculate the heart beat rate.

The pulse frequency is denoted by the letter f .

Heartbeat Sensor That Is Useful

Heart Rate Sensor is an example of a practical heartbeat sensor (Product No PC-3147). It's made up of an infrared led and a light-emitting diode (LED) enclosed in a clip-like structure. The detector part of the clip is attached to the skin and is attached to the organ (earlobe or finger).



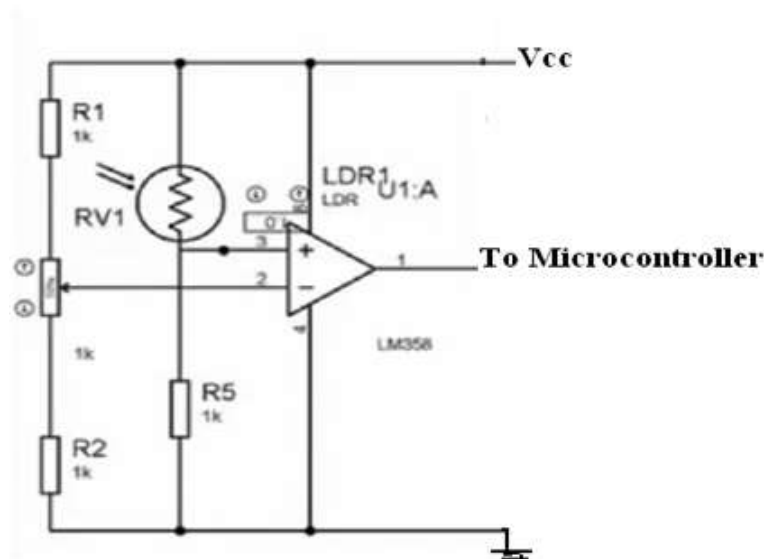
Fig no 8:Figure of Heart Beat Sensor

Another example is the TCRT1000, which has four pins: Pin 1 is used to provide power to the LED, while Pins 2 and 3 are grounded. The output is pin 4. Pin 1 is also the enable pin, and pulling it high activates the LED and activates the sensor. It's built into a wristband, and the output can be transmitted wirelessly (via Bluetooth) to a computer for processing.



Fig no 9:Application Creating a Heartbeat Sensor System from Scratch

Specific components such as a ldr, comparator IC LM358 and a Microcontroller may also be used to build a basic Heartbeat Sensor device, as seen below.



When the finger tissue or the earlobe tissue is illuminated using a light source, the light is modulated, i.e. a portion is absorbed by the blood and the remainder is released, as mentioned above for the concept of heart beat sensor. The light detector picks up this modulated light.

As a light detector, a Light Dependent Resistor (LDR) is used. It operates on the theory that as light shines on a resistor, it changes its resistance. The resistance reduces as the light intensity rises. As a result, the voltage drop around the resistor falls.

In this case, a comparator is used to equate the LDR output voltage to the threshold voltage. The voltage decrease around the LDR when a fixed-intensity light from the light source falls directly on it is known as the threshold voltage. The non-inverting terminal of the comparator LM358 is connected to the LDR, while the inverting terminal is connected to the potential divider arrangement set to the threshold voltage. When a light source is used to illuminate human skin, the strength of the light is reduced. The resistance increases as the decreased light pressure falls on the LDR, and the voltage decrease increases as a result. A logic high signal is produced at the comparator's output when the voltage drop over the LDR or non-inverting input

approaches that of the inverting input, and a logic low signal is generated when the voltage drop is smaller. As a result, the output consists of a sequence of pulses. These pulses may be fed to the Microcontroller, which then processes the data to determine the heart rhythm, which is then reflected on the Microcontroller's monitor.

TEMPERATURE SENSOR

The most often calculated environmental quantity is temperature. This is understandable given that temperature affects nearly all human, electrical, chemical, mechanical, and biological processes. Chemical reactions, biological systems, and even electrical circuits all perform well at specific temperatures. Temperature is one of the most widely measured quantities, so it's no surprise that there are a variety of methods for measuring it. Temperature sensing can be achieved in two ways: directly with the heating source, or indirectly using radiated energy instead of direct interaction with the source. Thermocouples, Resistance Temperature Detectors (RTDs), Thermistors, Infrared, and Semiconductor Sensors are only a few of the temperature sensors available today.

THERMOCOUPLE:

It's a kind of temperature sensor created by fusing two dissimilar metals together at one end. The HOT JUNCTION is the term for the joined end. The COLD END or COLD JUNCTION is the opposite end of these dissimilar metals. The cold junction is formed at the thermocouple's final point of contact. A minor voltage is produced when the temperature difference between the hot and cold junctions is large. This voltage is known as an EMF (electro-motive force), and it can be determined and used to determine temperature.

The RTD is a temperature sensing system that changes resistance as temperature rises. RTDs are mostly made of platinum, but they may also be made of nickel or copper. RTDs come in a variety of sizes, including wire wound and thin film. Apply a constant current to an RTD, calculate the corresponding voltage, and assess the RTD resistance. Over their operating areas, RTDs have a reasonably linear tolerance to temperature curves, and any nonlinearity is extremely stable and repeatable. Surface mount RTDs are used to calculate temperature on the PT100 RTD test platform. Temperature can also be measured in remote areas using an external

2, 3 or 4-wire PT100. A continuous current source is used to skew the RTDs. The current magnitude is relatively low to minimise self-heat caused by power dissipation. A reference voltage, one amplifier, and a PNP transistor are used in the constant current source circuit shown in figure.

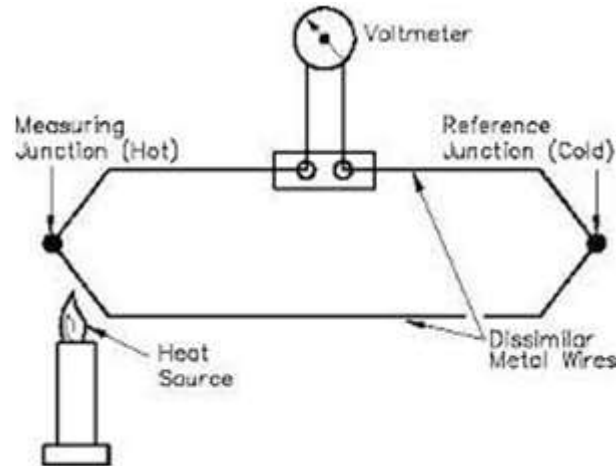


Fig no 10: Thermocouple

Thermistors:

The thermistor, like the RTD, is a temperature sensor whose resistance varies with temperature. Thermistors, on the other hand, are constructed from semiconductor materials. The resistance of thermistors is measured in the same way as that of RTDs, but their resistance vs. temperature curve is strongly nonlinear. As a result, we can see a significant resistance change with a very minor temperature change in the thermistors operating range. As a result, the device is extremely sensitive, making it suitable for set-point applications.

Semiconductor sensors:

Voltage from, current output, digital output, resistance output silicon, and diode temperature sensors are among the various categories. Over a temperature spectrum of around 55°C to +150°C, modern semiconductor temperature sensors have high precision and linearity. Internal amplifiers will scale the output to more practical values like 10mV/°C. They're also useful in cold-junction compensation circuits for thermocouples with a large temperature spectrum. This type of temperature sensor is described in more detail below.

SENSOR ICS

Temperature sensor ICs come in a multitude of shapes and sizes, allowing you to solve the widest range of temperature control problems. In a couple of key ways, these silicon temperature sensors vary from the previously described models. The first consideration is the operating temperature spectrum. A temperature sensor IC can run between -55°C and $+150^{\circ}\text{C}$, which is the nominal IC temperature range. The second big distinction is in terms of accessibility.

As an integrated circuit, a silicon temperature sensor may have comprehensive signal processing circuitry in the same box as the sensor. There are no compensation circuits needed for temperature sensor ICs. Some of them are analogue circuits that produce either voltage or current. Others provide alarm functions by combining analogue-sensing circuits with voltage comparators. Other sensor ICs integrate analogue sensing with digital input/output and control registers, making them suitable for microprocessor-based systems.

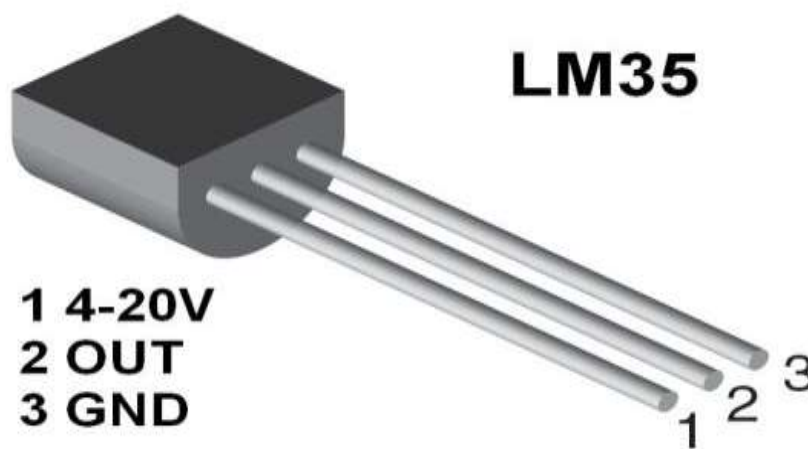


Fig no 11:Temperature Sensor

The above temperature sensor has three terminals and needs a maximum supply voltage of 5.5 V. This kind of sensor is made up of a material that changes resistance in response to temperature changes. The circuit detects the difference in resistance and measures the temperature. When the voltage is increased, the temperature rises as well. Using a diode, we may observe this activity.

Directly attached to the microprocessor input, temperature sensors are capable of direct and accurate communication with microprocessors. Without the use of A/D converters, the sensor unit can connect easily with low-cost processors.

The LM35 is an example of a temperature sensor. The LM35 series of temperature sensors are precision integrated-circuit temperature sensors with a linearly proportional output voltage to the Celsius temperature. The LM35 works between -55 and +120 degrees Celsius. The simple centigrade temperature sensor is depicted in the diagram below (+2 C to +150 C).

Features of LM35 Temperature Sensor:

- Calibrated in Celsius (Centigrade) and graded for the maximum range of -55 to +150 C. Suitable for remote applications.
- Due to wafer-level trimming, the cost is low.
- Adapts to voltages ranging from 4 to 30 volts
- Low levels of self temperature,
- Typical linearization of 1/4 C

Operation of LM35:

- The LM35 can be conveniently attached to other integrated circuit temperature sensors in the same way. It may be stuck or attached to a wall, and the temperature would be within 0.01 degrees Celsius of the surface temperature.
- This assumes that the ambient air temperature is close to the surface temperature; if the air temperature is either higher or lower than the surface temperature, the real temperature of the LM35 die will be somewhere in the centre.

Pressure sensor

A pressure sensor is a device that detects the pressure of gases or liquids. Pressure is the force used to prevent a fluid from spreading, and it is normally expressed in force per unit area. A pressure sensor is typically used as a transducer, producing a signal as a result of the applied pressure. Such a signal is electrical for the purposes of this document.

Pressure sensors are found in thousands of daily applications for tracking and monitoring. Other variables such as fluid/gas flow, speed, water level, and altitude can be measured indirectly using pressure sensors. Other terms for pressure sensors include pressure transducers, pressure transmitters, pressure senders, pressure detectors, piezometers, and manometers.

Pressure sensors differ greatly in terms of technology, architecture, efficiency, applicability, and expense. Pressure sensors are made by over 50 inventions and at least 300 firms around the world, according to an optimistic survey.

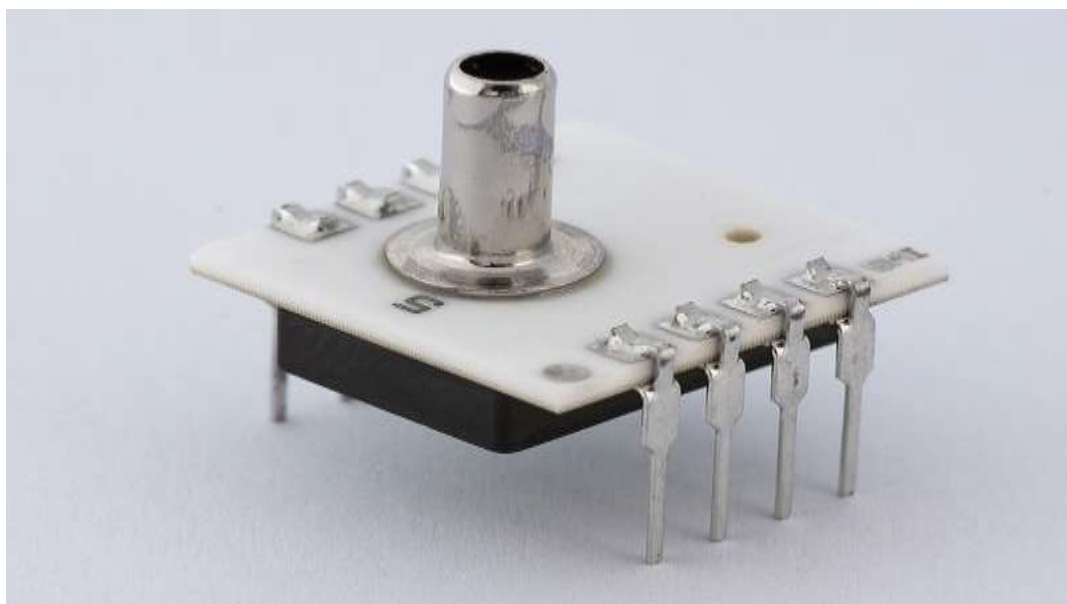


Fig No 12:pressure Sensor

IOT:

The following is the chapter's structure: The motive for this project is described in Section 1.1. The study priorities and concerns are mentioned in Section 1.2. Section 1.3 outlines the actions necessary to accomplish our goals. The report's form as seen in Section 1.4.

1.1 Background

The Future Web objective is to give a framework to have a quick admittance to data about the actual world and its articles. Actual items can be pertinent to various application spaces, for example, e-wellbeing, distribution center administration, and

so forth Every application area may have various sorts of actual gadgets. Each actual gadget can have its own details, which is needed to use to cooperate with it. To accomplish the future Web objective, a layered vision is required that can work with information access. Web of Things (IoT) is a dream that plans to incorporate the virtual universe of data to this present reality of gadgets through a layered engineering.

The expression "Web of Things" comprises of two words, in particular Web and Things. Web alludes to the worldwide organization foundation with versatile, configurable abilities dependent on interoperable and standard correspondence conventions. Things are actual items or gadgets, or virtual articles, gadgets or data, which have characters, actual ascribes and virtual characters, and utilize insightful interfaces [1]. For example, a virtual item can address a theoretical unit of sensor hubs that contains metadata to recognize and find its comparing sensor hubs. In this way, IoT alludes to the things that can give data from the actual climate through the Web.

Middleware is as an interface between the equipment layer and the application layer, which is liable for cooperating with gadgets and data the board [2]. The part of a middleware is to introduce a bound together programming model to cooperate with gadgets. A middleware is responsible for concealing the heterogeneity and appropriation issues that we face while interfacing with gadgets [3].

1.2 Motivation

IoT-based framework is responsible for giving information from a climate to a non-master client. IoT-based framework can be utilized in various conditions, so it should have the option to address numerous heterogeneous gadgets. Consequently, a significant worry inside building up an IoT-based framework is the means by which to deal with the cooperation with the heterogeneous gadgets for non-master clients. This worry can be tended to by a middleware layer among gadgets and non-master clients. This layer is dependable to conceal the variety of gadgets from the client viewpoint, and gives access straightforwardness to the gadgets for the end clients.

Making deliberations of gadgets been tended to in the writing. The middleware we found in the writing can furnish fulfillment by working with the communication with gadgets, yet they don't uphold low-level gadget design [4].

2.1. IoT definition

In this section, we explain some of the IoT definitions. Also, we explain the layered architecture for IoT.

Internet of Things (IoT) has increasingly gained attention in industry to interact with different types of devices. IoT can have influence on industry and society by integrating physical devices into information networks [8]. IoT impacts can be on different perspectives, namely for private and business users.

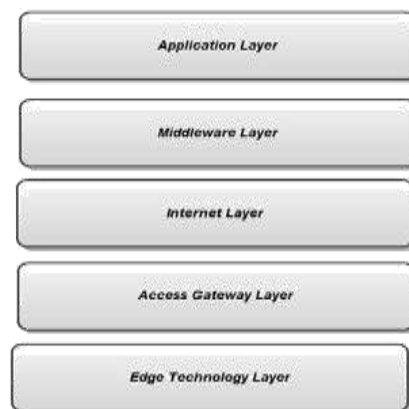


Figure 13: Layered architecture of the Internet of Things (IoT).

2.2 Challenges

Based frameworks can be utilized for various purposes and territories, so that, we need to confront various difficulties. In this part, we clarify a portion of the provokes that should be considered in research exercises [10]:

Edge innovation

At the equipment level, more exploration endeavors are needed to build up the innovation of installed gadgets, sensors, actuators and (aloof and dynamic) distinguishing proof, since an IoT-based framework should have the option to accumulate adequate data about this present reality by utilizing an enormous assortment of gadgets and conditions. In this manner, there more work is as yet

needed to associate heterogeneous gadgets and convey them in IoT applications, and to offer help for new gadgets.

Networking advancements

In IoT, things are associated through various types of organizations, for example portable, wired and remote organization. These organizations support bi-directional correspondence at various levels among this present reality articles, applications and administrations that are utilized by the IoT applications. This exceptionally dispersed construction ought to give interconnection low energy utilization, while conveyed information can cause security issues.

Middleware framework

In IoT, we have heterogeneous gadgets and organizations. Their heterogeneity can conceivably increment with new advances. To work with the utilization of these gadgets by IoT applications, we should protect their heterogeneity. Consequently, we need to build up a protected, adaptable and semantically enhanced help arranged middleware to adapt to the heterogeneity of gadgets.

Platform administrations

They support an unrivaled administration of all elaborate gadgets in an incorporated manner, guaranteeing versatility, high accessibility, and the free from any danger execution of the mentioned usefulness from gadgets. In proceed, we center around the middleware challenges, since we are searching for the functionalities that a middleware can accommodate the application layer in the IoT-based frameworks.

2.3 IoT_based Middleware

Bandyopadhyay, S. et. al. have studied the middleware systems that have been applied in IoT-based systems [1]. They classify the required functionality of middleware to manage interaction with a variety of devices in four functional components, namely (1) interface protocols, (2) device abstraction, (3) central control, context detection & management, and (4) application abstraction (shown in Fig. 2.2). In the following, we explain these components in details.

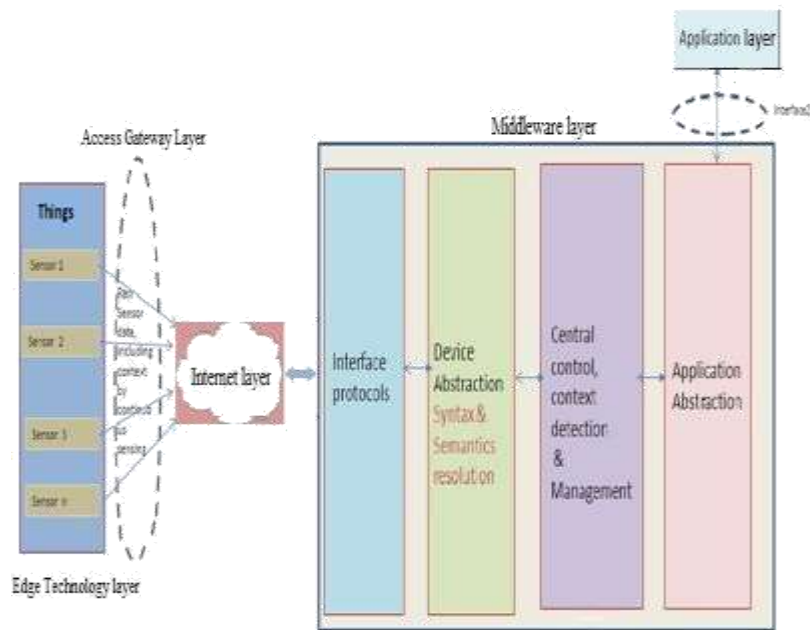


Figure 14: Functional components of a middleware for IoT-based systems

2.3.1 Interface protocols

This segment is responsible for giving specialized interoperability. Interoperability with regards to Interface conventions implies: the capacity of two frameworks to interoperate by utilizing similar correspondence conventions. As per ETSI (European Broadcast communications Principles Establishment) [14] specialized interoperability is characterized as the relationship of equipment or programming parts, frameworks and stages that empower machine-to-machine correspondence to happen. This sort of interoperability is frequently fixated on (correspondence) conventions and the framework required for those conventions to operate[14].

The Interface Convention part characterizes conventions for trading data among various organizations that may work dependent on various correspondence conventions, to permit specialized interoperability. This part is answerable for taking care of fundamental availability in the physical and information interface, organization, transport, and in some cases the application layer of the TCP/IP stack. To adapt to the heterogeneity of gadgets, we can utilize a covering for every gadget to interpret the convention upheld by the gadget to a typical convention. This covering can be set either on the gadget side or the middleware side. On the off chance that we need to have direct association with gadgets, we should put the covering in the middleware

side. Gadgets generally have restricted ability of computational interaction, so this would be motivation to execute covering on the middleware side. Conversely, if there should be an occurrence of backhanded connection with gadgets we can build up a delegate covering between the middleware and the gadgets. The interface convention segment is mindful to permit the middleware to help both immediate and aberrant associations.

2.3.2 Device Abstraction (DA)

This segment is answerable for giving a theoretical arrangement to work with the association of the application parts with gadgets. This reflection gives syntactic and semantic interoperability, which are characterized by ETSI [14] as follows:

- Syntactic interoperability is related with information designs. The messages moved by correspondence conventions should have an all around characterized grammar and encoding design, which can be addressed by utilizing undeniable level exchange linguistic uses, for example, HTML and XML.
- Semantic interoperability is typically connected with the significance of the substance of message which is reasonable for human. Hence, interoperability on this level implies that there is a typical comprehension among individuals on the importance of the substance (data) being traded among them. Be that as it may, since DA doesn't discuss straightforwardly with human, semantic interoperability with regards to DA is responsible for giving this basic comprehension to applications.

Semantic interoperability depends on semantic models which will in general be space explicit. For instance, one approach to give semantic interoperability in Help Arranged (SOA) [15] based middleware is by utilizing Gadgets Profile for Web Administrations (DPWS) [16] . In this specific circumstance, every gadget type alludes to a recognized assistance type DA segment give two general functionalities: (1) to request that gadgets play out some usefulness and (2) to characterize and design gadgets DPWS utilizes the XML design that is appeared in Code 2.1.

```

<dpws:ThisModel>
  <dpws:Manufacturer>ACME Manufacturing</dpws:Manufacturer>
  <dpws:ModelName xml:lang="en-GB" >
    ColourBeam 9
  </dpws:ModelName>
  <dpws:ModelName xml:lang="en-US" >
    ColorBeam 9
  </dpws:ModelName>
</dpws:ThisModel>
</wsx:MetadataSection>
<wsx:MetadataSection
  dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/ThisDevice"
>
  <dpws:ThisDevice>
    <dpws:FriendlyName xml:lang="en-GB" >
      ACME ColourBeam Printer
    </dpws:FriendlyName>
    <dpws:FriendlyName xml:lang="en-US" >
      ACME ColorBeam Printer
    </dpws:FriendlyName>
  </dpws:ThisDevice>

```

Code 2.1 XML code to define a device

2.3.3 Central control, Context detection & Management (CCM)

Setting portrays the circumstance of a substance, which can be a spot, an individual or an article that is pertinent to the client, applications and their associations [1]. The CCM useful part is mindful to help setting mindful calculation that is a computational style that consider the setting of the elements that collaborate with the framework. A middleware for IoT-based frameworks should be setting mindful to work in brilliant conditions [1]. Savvy conditions allude to an actual world that is luxuriously and undetectably joined with sensors, actuators, shows, and computational components, implanted flawlessly in the ordinary objects of our lives, and associated through a ceaseless organization [17]. Setting mindfulness incorporates two functionalities:

- 1) Context recognition, which comprises of gathering information from assets, and choosing the data that can affect the calculation.
- 2) Context handling, to utilize the assembled data to play out an assignment or settle on a choice.

Application Abstraction

This useful part gives an interface to both undeniable level applications and

end-clients to cooperate with gadgets. For example, this interface can be a Soothing interface or can be carried out with some question based language.

A. Malatras. et. al. propose a SOA-based middleware to perform information the executives and information observing administrations [18]. This middleware utilizes a Soothing interface to work with the cooperation of undeniable level applications with sensors, which can speak with Remote Sensor Organization (WSN) through the HTTP activities: (1) Will give an inquiry on a current asset, (2) Erase to eliminate a current asset, (3) POST to make another asset, and (4) PUT to refresh a current asset. For example, a customer application gets the consequence of a space task by sending a GET demand through the URL: `http :/{hostname}/REST/{version}/DomaintaskResult/id'`. In this URI that reference to a „id“ prompts a special outcome.

In the TinyDB middleware [19], an end-client can utilize a question language to associate with gadgets. For example, a client can send the accompanying question to get the id of a sensor (nodeid) close by and temperatures that are detected by this sensor during the previous 10 seconds prior to executing the inquiry:

```
SELECT nodeid, temp
FROM sensors
Test PERIOD 1s FOR 10s
```

2.4 End

In this part, we characterized IoT and distinguished normal IoT layers. Besides, we examined a reference middleware design for IoT-based frameworks. This engineering has been proposed by Bandyopadhyay, S. et. al dependent on an examination on the current middleware structures for IoT-based frameworks [1].

ARDUINO UNO AND ITS PROGRAMMING

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing

platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be communicate with software running on your computer. The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

2.1 Overview

The Arduino microcontroller is an easy to use yet powerful single board computer that has gained considerable traction in the hobby and professional market. The Arduino is open-source, which means hardware is reasonably priced and development software is free. This guide is for students in ME 2011, or students anywhere who are confronting the Arduino for the first time. For advanced Arduino users, prowl the web; there are lots of resources.

This guide covers the Arduino Uno board (Spark fun DEV-09950, \$29.95), a good choice for students and educators. With the Arduino board, you can write programs and create interface circuits to read switches and other sensors, and to control motors and lights with very little effort. Many of the pictures and drawings in this guide were taken from the documentation on the

This is what the Arduino ide looks like.



The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++.[2] It is used to write and upload programs to Arduino compatible boards, but also, with the help of third-party cores, other vendor development boards.[3]

The source code for the IDE is released under the GNU General Public License, version 2.[4] The Arduino IDE supports the languages C and C++ using special rules of code structuring.[5] The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub `main()` into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution.[6] The Arduino IDE employs the program `avrdude` to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.[7] By default, `avrdude` is used as the uploading tool to flash the user code onto official Arduino boards.[8]

Arduino Pro IDE

[Developer\(s\)](#)

Arduino Software

Preview release	v0.1.2 / 14 September 2020; 4 months ago ^[9]
Repository	github.com/arduino/Arduino 
Written in	C , C++
Operating system	Windows , macOS , Linux
Platform	IA-32 , x86-64 , ARM
Type	Integrated development environment
License	LGPL or GPL license
Website	blog.arduino.cc/2020/08/24/cli-and-ide-get-better-together/

With the rising popularity of Arduino as a software platform, other vendors started to implement custom open source compilers and tools (cores) that can build and upload sketches to other microcontrollers that are not supported by Arduino's official line of microcontrollers.

In October 2019 the Arduino organization began providing early access to a new Arduino Pro IDE with debugging^[10] and other advanced features.^[11]

The Duemilanove board features an Atmel ATmega328 microcontroller operating at 5 V with 2 Kb of RAM, 32 Kb of flash memory for storing programs and 1 Kb of EEPROM for storing parameters. The clock speed is 16 MHz, which translates to about executing about 300,000 lines of C source code per second. The board has 14 digital I/O pins and 6 analog input pins. There is a USB connector for talking to the host computer and a DC power jack for connecting an external 6-20 V power source, for example a 9 V battery, when running a program while not connected to the host

computer. Headers are provided for interfacing to the I/O pins using 22 g solid wire or header connectors.

The Arduino programming language is a simplified version of C/C++. If you know C, programming the Arduino will be familiar. If you do not know C, no need to worry as only a few commands are needed to perform useful functions.

2.3 Pin Configuration

The NodeMCU can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V. this pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current

draw is 50 mA.

- GND. Ground pins.
- IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM.

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using `pinMode()`, `digital Write()`, and `digital Read()` functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kohms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and Transmit (TX) TTL serial data.

These pins are connected to the corresponding pins of the ATmega8U2 USB-to- TTL Serial chip.

- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the `attachInterrupt()` function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the `analog Write()` function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is

HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analog Reference () function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the Wire library.

There are a couple of other pins on the board:

AREF. Reference voltage for the analog inputs. Used with analog Reference ().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

2.4 Communication

Microcontrollers depend on a host computer for developing and compiling programs. The software used on the host computer is known as an integrated development environment, or IDE. For the Arduino, the development environment is based on the open source Processing platform (www.processing.org) which is described by its creators as a “programming language and environment for people who want to program images, animation, and interactions.” The Arduino programming language leverages an open source project known as Wiring (wiring.org.co). The Arduino language is based on good old- fashioned C. If you are unfamiliar with this language, don't worry; it's not hard to learn, and the Arduino IDE provides some feedback when you make mistakes in your programs.

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears

as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A Software Serial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

As you go through the list of programming statements available in the Arduino IDE (choose Help->Reference), you might think there isn't much power for doing things like running servos, operating stepper motors, reading potentiometers, or displaying text on an LCD. Like most any language based on C, the Arduino supports the notion of "libraries" code

Repositories that extend core programming functionality. Libraries let you re-use code without having to physically copy and paste it into all your programs. The standard Arduino software installation comes with several libraries you may use, and you can download others from the Arduino support pages and from third-party websites that publish Arduino library code. A good example of a library you'll use with the Robot and likely many other robot projects is

Servo. This library allows you to connect one or more hobby R/C servos to the Arduino's digital I/O pins. The Servo library comes with the standard Arduino installation package Library->Servo. This adds the line

```
#include <Servo.h>
```

Which tells the Arduino IDE that you wish to include the Servo library in your sketch. With the functionality of the library now available to you, you can use its various functions to control one or more servos. For example, you can use the write function

to rotate a servo to a specific position, from 0 to 180 degrees. The following code

```
myServo.write(90);
```

Moves a servo to its midpoint, or 90 degree position. Structurally, Arduino sketches are very straightforward and are pretty easy to read and understand. The Arduino program contains two main parts: `setup ()` and `loop ()`. These are programming functions that do what their names suggest: `setup ()` sets up the Arduino hardware, such as specifying which I/O lines you plan to use, and whether

They are inputs or outputs. The `loop ()` function is repeated endlessly when the Arduino is operating.

Arduino IDE (Integrated development environment) is used to write the program and dump into the Arduino board

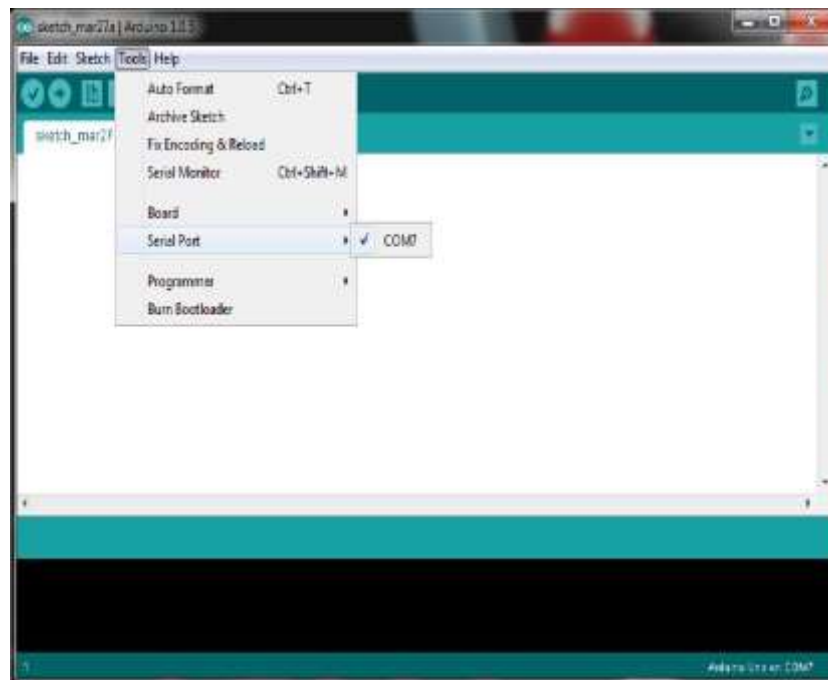
2.5 ARDUINO SOFTWARE:

1. Open Arduino IDE as shown below

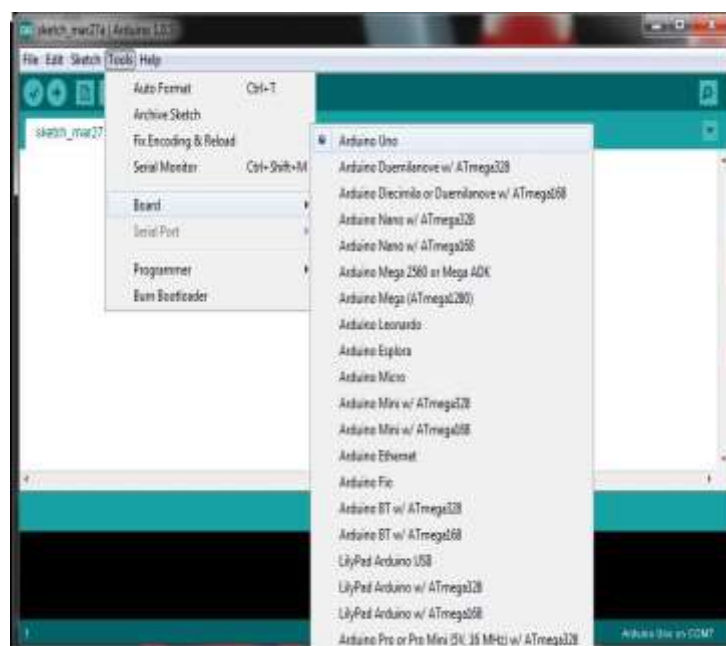
Open Arduino IDE



2. Select the COM Port from tool Select the COM Port



3. Select the required Arduino board from Tools as shown below
Select the required Arduino board



4. Write the sketch in Arduino IDE
Sketch in Arduino IDE

```

sketch_mar28a
File Edit Sketch Tools Help

#include <Servo.h> //includes the servo library
int motor_pin1 = 4;
int motor_pin2 = 5;
int motor_pin3 = 6;
int motor_pin4 = 7;
int servopin = 8;
int sensorpin = 0;
int dist = 0;
int leftdist = 0;
int rightdist = 0;
int object = 500; //distance at which the robot should look for another route
const int RightSensor = 1; // This pin is used to read the value of the Right Sensor.
const int LeftSensor = 2; // This pin is used to read the value of the Left Sensor.
int SensorLeft; // This stores the value of the Left Sensor pin to use later on in the sketch.
int SensorRight; // This stores the value of the Right Sensor pin to use later on in the sketch.
int SensorDifference; // This value is used to determine the difference between the Left and Right

```

5. Compile and upload the Sketch to Arduino board upload the Sketch to Arduino board

```

sketch_mar28a
File Edit Sketch Tools Help

#include <Servo.h> //includes the servo library
int motor_pin1 = 4;
int motor_pin2 = 5;
int motor_pin3 = 6;
int motor_pin4 = 7;
int servopin = 8;
int sensorpin = 0;
int dist = 0;
int leftdist = 0;
int rightdist = 0;
int object = 500; //distance at which the robot should look for another route
const int RightSensor = 1; // This pin is used to read the value of the Right Sensor.
const int LeftSensor = 2; // This pin is used to read the value of the Left Sensor.
int SensorLeft; // This stores the value of the Left Sensor pin to use later on in the sketch.
int SensorRight; // This stores the value of the Right Sensor pin to use later on in the sketch.
int SensorDifference; // This value is used to determine the difference between the Left and Right

```

Compiling sketch.

UBIDOTS EXPLORER:

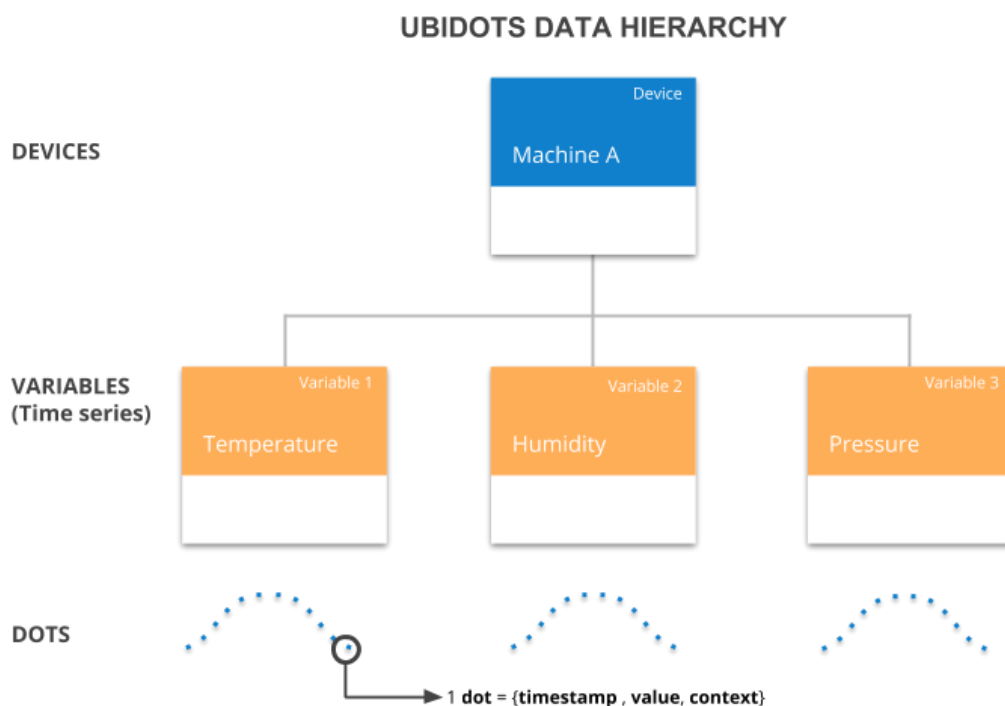
Introduction

Welcome to the Ubidots REST API hardware docs. Here you would reference how to send or retrieve data from your hardware devices using our supported communication protocols: HTTP, MQTT and TCP/UDP.

The purpose of this section is to help you understand what happens in the backstage when communicating with Ubidots, so you can replicate this in your firmware. For this reason, we avoid the use of examples using libraries. However, if you'd like to use one of our libraries (Python, C, and more), check out our compatible devices and API clients sections.

How Ubidots Works?

Every time a device updates a sensor value in a variable, a data-point or "dot" is created. Ubidots stores dots that come from your devices inside variables, and these stored dots have corresponding timestamps:



Each dot contains these items:

Item	Description	Mandatory
value	A numerical value. Ubidots accepts up to 16 floating-point length numbers.	YES
timestamp	Unix Epoch time, in milliseconds. If not specified, then our servers will assign one upon reception.	NO
context	An arbitrary collection of key-value pairs. Mostly used to store the latitude and longitude coordinates of GPS devices.	NO

Values

A numerical value. Ubidots accepts up to 16 floating-point length numbers.

```
{"value" : 34.87654974}
```

Timestamps

A **timestamp**, as best described here, is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Therefore, the unix time stamp is merely the number of seconds between a particular date and the Unix Epoch. Please keep in mind that when you send data to Ubidots, you must set the timestamp in milliseconds; also, if you retrieve a dot's timestamp, it will be in milliseconds.

```
"timestamp" : 1537453824000
```

The above timestamp corresponds to Thursday, September 20, 2018 2:30:24 PM.

PRO-TIP: A useful tool to convert between Unix timestamps and human-readable dates is Epoch Converter.

Context

Numerical values are not the only data type supported; you can also store string or char data types inside what we call context. The **context** is a key-value object that allows you to store not only numerical but also string values. An example use of the

context could be:

```
"context" : {"status" : "on", "weather" : "sunny"}
```

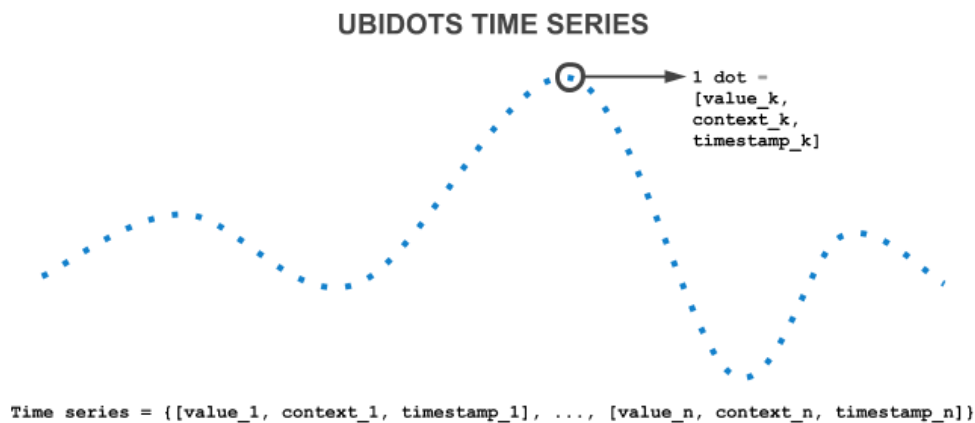
A **context** is commonly used to store the latitude and longitude coordinates of your device for GPS/tracking application use cases. All Ubidots maps uses the lat and lng keys from a dot's **context** to extract the coordinates of your device, in that way you just need to send a single dot with the coordinates values in the variable context to plot a map instead of sending separately both latitude and longitude in two different variables. Below you can find a typical **context** with coordinates values:

```
"context" : {"lat":-6.2, "lng":75.4, "weather" : "sunny"}
```

Please note that you can mix both string and numerical values in the context. If your application is for geo-localization purposes, make sure that the coordinates are set in decimal degrees.

Time Series

Based on the above, we can illustrate an Ubidots time series like this:



Ubidots is an agnostic platform, this means that it does not really care what hardware device you are using, as long as you're able to interact with us through at least one of these protocols:

- HTTP
- MQTT
- TCP/UDP

HTTP

HTTP is the main internet protocol for information exchange in the Internet. It also supports data encryption, which is usually called secure HTTP or HTTPS. When you use a web browser (MS Edge, Firefox, Google Chrome, etc) you are sending and receiving packages using HTTP or HTTPS. Because of this, it was not surprising to see HTTPS become a popular option for data exchange within IoT.

When a developer is to implement HTTP communication, then he should look for a RESTful Application Programming Interface, or REST API, which exposes all the endpoints required to interact with a third party application (like Ubidots). In this section, you can reference our REST API to understand how to send and retrieve data from our servers.

Note: Our goal with this section is to help you understand what's happening behind when communicating with Ubidots, enabling you to replicate it within your own firmware. Because of this, we avoid using our custom libraries in all the examples.

HTTP requests

The following methods are specified within the HTTP standard:

HTTP Method	Description
GET	Used to retrieve information
POST	Used to create a new element
PATCH	Used to update existing elements
DELETE	Used to delete existing elements

HTTP is a **request/response** protocol, this means that every request that you make is answered by the server. This response includes a number (response code)

and a body. For example, when you make a request to retrieve a file on a webpage "(e.g. "Get me the file 'website.html'"), you build a GET request. If the request is correct, the server will typically return a 200 response code, along with the file requested (body).

Under the same logic, when a device sends a GET request to the Ubidots' server (i.e. requesting the last value of a variable), then the server sends back a response with the status of the request (response code), and a body, which would be the a value with its context and timestamp.

An HTTP request also needs the parameters below:

- Host: Specifies the server you will be making HTTP requests to.
- Path: This is typically the remaining portion of the URL that specifies the resource you want to consume, be it a variable or a device. For example, if an API endpoint is: `industrial.api.ubidots.com/api/v1.6/devices/my-device` then the path would be `/api/v1.6/devices/my-device`
- Headers: Define the operating parameters of the HTTP request such as authentication, Content-Type, Content-Length, etc.
- Body/payload: In the case of POST and PATCH requests, this is the data sent by your device to the server. GET requests typically do not have a body because they are meant to request data, not to send data.

Ubidots accepts data as JavaScript Object Notation or JSON. JSON is a typical HTTP data type, it is a collection of name/value pairs. In various programming languages, this is treated as an object, record, struct, dictionary, hash table, keyed list, or associative array. It is also human readable and language independent. An example of a JSON data type that Ubidots accepts can be referenced below:

```
{"temperature": {"value":10, "timestamp": 1534881387000, "context": {"machine": "1st floor"}}}
```

A typical HTTP request to Ubidots should be set as below:

```
POST {PATH} HTTP/1.1<CR><LN>
```

Host: {HOST}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
Where:

- {PATH}: Path to the resource to consume
Example: /api/v1.6/variables/ to get user's variables information.
- {HOST}: Host URL.
Example: industrial.api.ubidots.com
- {USER_AGENT}: An optional string used to identify the type of client, be it by application type, operating system, software vendor or software version of the requesting user agent.
Examples: ESP8266/1.0, Particle/1.2
- {TOKEN}: Unique key that authorizes your device to ingest data inside your Ubidots account.
- {PAYLOAD_LENGTH}: The number of characters of your payload.
Example: The payload {"temperature": 20} will have a content-length of 19.
- {PAYLOAD}: Data to send.
Example: {"temperature": 20}

PRO TIP: The easy way of handle HTTP and HTTPs requests is by using cURL, a command line tool. To learn how to install cURL on Windows, MacOSX or Linux please refer to this guide.

API URLs

API access can be over HTTP or secure HTTP, HTTPs, using the following endpoints based on your Ubidots Account type.

Security Note: We strongly advise to use HTTPs to make sure your data travels

encrypted, avoiding the exposure of your API token and/or sensor data.

HTTP

Ubidots Account	Endpoint	Port
Educational	http://things.ubidots.com	80
Industrial	http://industrial.api.ubidots.com	80

HTTPSs

Ubidots Account	Endpoint	Port
Educational	https://things.ubidots.com	443
Industrial	https://industrial.api.ubidots.com	443

Ubidots supports SSL v1.1, TLS v1.2 and v1.3. You can download our root certificates in different formats:

- PEM file: A certificate chain with two root certificates from our certificate authorities (CAs).
- DER file: Same as the PEM file, with an alternative encoding.
- CRT file: Same as the PEM file, with a different extension. Often referred to as .crt, .cert or .cer.

NOTE: To avoid future inconveniences, we advise to **use both certificates** in your device's auth routines. While the second is the current valid certificate, we will use the first after 2021.

Fixed IPs

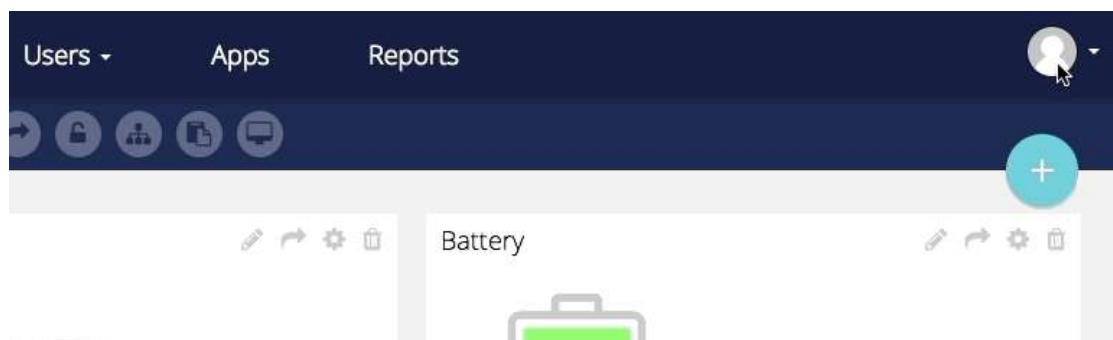
Ubidots also supports fixed IPs address for data ingestion, but we highly advise to our users to implement their firmware routines using the dns endpoints instead of fixed IPs, as these last ones may change in the future.

Ubidots Account	ipv4	ipv6	Port
Educational	50.23.124.68	2607:f0d0:2101:39::2	80/443
Industrial	169.55.61.243	2607:f0d0:3:b::2	80/443

HTTP Authentication

Every request requires a token. A token is an unique key that authorizes your device to ingest data inside your Ubidots account.

The easiest way to get yours is clicking on "API Credentials" under your profile tab:



Also, 6 hour valid tokens can be generated using your account API-KEY. If you desire to learn how to do it, please refer the Ubidots REST Software API reference.

There are two ways to send a token in a request:

1. Send the token as parameter in the endpoint, setting it with the `?token=` expression. This is the easiest one to implement but the most unsecure method to send your authentication credentials.
2. Send the token as a request header. For this, use the X-Auth-Token header.

Security Note: While sending your token as a URL parameter can be quite comfortable in some devices, we only advise to do this for prototyping stages. For production stages, we strongly advise sending the token using the X-Auth-Token header.

GET

<https://industrial.api.ubidots.com/api/>

Send data

Our REST API allows you send data to the platform in two different ways:

- Send data to a **device**
- Send data to a **variable**

If you're looking for more advanced API capabilities, such as editing a device name, or listing all of your devices, please refer to our Ubidots Software REST API Reference.

Send data to a device

Request structure:

```
POST /api/v1.6/devices/{DEVICE_LABEL} HTTP/1.1<CR><LN>
Host: {Host}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
```

Expected Response:

```
HTTP/1.1 200 OK<CR><LN>
Server: nginx<CR><LN>
Date: Tue, 04 Sep 2018 22:35:06 GMT<CR><LN>
Content-Type: application/json<CR><LN>
Transfer-Encoding: chunked<CR><LN>
Vary: Cookie<CR><LN>
Allow: GET, POST, HEAD, OPTIONS<CR><LN><CR><LN>
{PAYLOAD_LENGTH_IN_HEXADECIMAL}<CR><LN>
{"{VARIABLE_LABEL}": [{"status_code": 201}]}<CR><LN>
0<CR><LN>
```

The easiest way to **send values to Ubidots** is by specifying your Device label in the request path and making a POST request to it.

Replace the {DEVICE_LABEL} with the unique label of the desired device to send data to. If the device does not exist in your Ubidots account, it will be automatically created. Device labels can contain any alphanumerical character, without blank spaces, i.e weather-station, my-home, esp8266.

To set the variables values, just create a JSON data payload as below

Nº Variables	JSON structure
One	<code>{"variable-label": VALUE}</code>
Multiple	<code>{"variable-label-1": {"value": VALUE, "timestamp": TIMESTAMP, "context": {"key": VALUE}}, ... , "variable-label-n": {"value": VALUE, "timestamp": TIMESTAMP, "context": {"key": VALUE}}}</code>

timestamp and context parameters are optional.

NOTE 1: By default, the Ubidots backend will take the latitude and longitude values inside any variable labeled as "gps", "location", position or "coordinates" automatically to get your device coordinates in all of the map widgets. Alternatively, you change the default location variable.

NOTE 2: All timestamps must be in milliseconds. If you set a timestamp for a variable value, it will be stored with the date represented by it. You can easily convert dates to timestamps values [here](#).

Examples

- Single dot: `{"temperature": 27}`
- Multiple dots: `{"temperature": 27, "humidity": 55, ... , "pressure": 78}`

Including additional keys like context and timestamp

- Single dot: `{"position": {"value" : 1, "timestamp": 1514808000000,`

"context":{"lat":-6.2, "lng":75.4, "name" : "John"}}}

- Multiple dots: {"temperature": {"value": 27, "timestamp": 1514808000000}, "humidity": 55, "pressure": {"value": 78, "context":{"name" : "John"}}}

POST

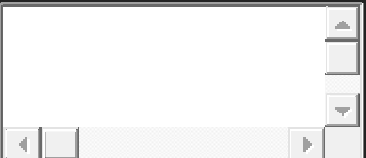
https://industrial.api.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}

POST

https://things.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}

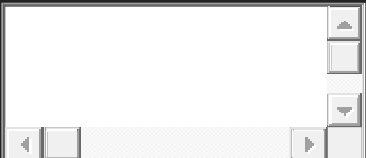
Send a single dot (Educational users)

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"temperature": 27}' https://things.ubidots.com/api/v1.6/devices/my-new-device
```



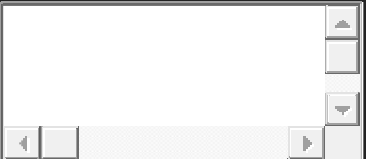
Send a single dot (Industrial users)

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"temperature": 27}' https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device
```



Send a dot with coordinates

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"position": {"value" : 1, "timestamp": 1514808000000, "context":{"lat":-6.2, "lng":75.4}}}' https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device
```



Send multiple dots

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"temperature": {"value": 27, "timestamp": 1514808000000}, "humidity": 55, "pressure": {"value": 78, "context":{"name" : "John"}}}' https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device
```



Send data to a variable

You can also send one or multiple dots to a single variable.

Request structure:

```
POST /api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values
HTTP/1.1<CR><LN>
Host: {Host}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
```

Expected Response:

```
HTTP/1.1 201 Created<CR><LN>
Server: nginx<CR><LN>
Date: Tue, 04 Sep 2018 22:50:55 GMT<CR><LN>
Content-Type: application/json<CR><LN>
Transfer-Encoding: chunked<CR><LN>
Vary: Accept, Cookie<CR><LN>
Location: http://industrial.api.ubidots.com/api/v1.6/values/{VARIABLE_ID}<CR><LN>
Allow: GET, POST, HEAD, OPTIONS<CR><LN><CR><LN>
{PAYLOAD_LENGTH_IN_HEXADECIMAL}<CR><LN>
```

```
{"url": "http://things.ubidots.com/api/v1.6/values/{VARIABLE_ID}", "value": 27.0,
"timestamp": 1536101455822, "context": {}, "created_at": "2018-09-04T22:50:55.822035Z"}<CR><LN>
```

0<CR><LN>

In the path specified above, replace the {DEVICE_LABEL} and {VARIABLE_LABEL} keys with the unique labels of the Ubidots device and variable you wish to send data to. If the specified device and/or variable do not exist, they will be automatically created.

Examples

- Single dot: {"value":27}
- Single dot with context and timestamp: {"value":27, "timestamp": 1514808000000, "context":{"lat":37.773, "lng":-122.431}}
- Multiple dots (must include the timestamp of each value): [{"value": 27, "timestamp":1514808000000}, {"value": 12, "timestamp":1514808900000}]

Send a single dot (Educational users)

POST

```
https://industrial.api.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values
```

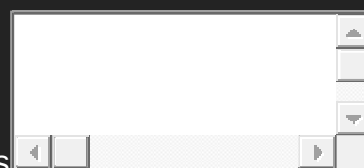
POST

```
https://things.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values
```

Send a single dot (Industrial users)

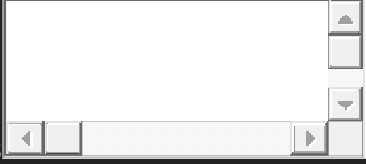
```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"value": 27}' https://industrial.api.ubidots.com/api/
```

```
v1.6/devices/my-new-device/my-variable/values
```



Send a single dot (Educational users)

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"value": 27}' https://things.ubidots.com/api/v1.6/devices/my-new-device/my-variable/values
```



Send multiple dots

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '[{"value": 27, "timestamp":1514808000000}, {"value": 12, "timestamp":1514808900000}]' https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device/my-variable/values
```



Retrieve data

The Ubidots REST API allows you to retrieve data from the platform data base. A typical application for data retrieval is actuators like motors or lights control.

Retrieve mutiple values from a variable

Request structure:

```
GET /api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values HTTP/1.1<CR>><LN>
```

```
Host: {Host}<CR><LN>
```

```
User-Agent: {USER_AGENT}<CR><LN>
```

```
X-Auth-Token: {TOKEN}<CR><LN>
```

```
Content-Type: application/json<CR><LN><CR><LN>
```

Expected Response:

```
HTTP/1.1 200 OK<CR><LN>
```

```
Server: nginx<CR><LN>
```

```
Date: Tue, 04 Sep 2018 22:35:06 GMT<CR><LN>
Content-Type: application/json<CR><LN>
Transfer-Encoding: chunked<CR><LN>
Vary: Cookie<CR><LN>
Allow: GET, POST, HEAD, OPTIONS<CR><LN><CR><LN>
{PAYLOAD_LENGTH_IN_HEXADECIMAL}<CR><LN>
{"count": true, "previous": null, "results": [{PAYLOAD_WITH_VALUES}], "next": {URL
_WITH_ADDITIONAL_VALUES}}<CR><LN>
0<CR><LN>
```

In the path specified above, replace the {DEVICELABEL} and {VARIABLE_LABEL} keys with the unique labels of the Ubidots device and variable you desired to retrieve data from. This `_path` returns by default the last 50 values stored of your variable. You can manage the number of dots to retrieve using the **page_size** parameter in your path:

```
/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values/?page_size=2
```

The path above retrieves the last 2 dots of your variable.

Ubidots API also allows you to filter the values to be retrieved by date range, number of dots, and more. If you desire to go deep into this kind of requests, please check the Ubidots Software REST API Reference.

```
GET
https://industrial.api.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values
GET
https://things.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/values
```

Get the last two values from a variable (Educational users)

```
curl -X GET -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" https://things.ubidots.com/api/v1.6/devices/my-device/
```

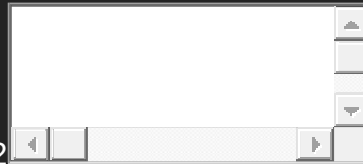
my-variable/values/?page_size=2



Get the last two values from a variable (Industrial users)

```
curl -X GET -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" https://industrial.api.ubidots.com/api/v1.6/devices/my-d
```

evice/my-variable/values/?page_size=2



Retrieve the last value from a variable

A typical control application in general only needs to ask for the last value of a variable, i.e, if you push a virtual switch at Ubidots, you just need to know what is the last value of the variable related with that switch to take actions on any actuators at your side.

Request structure:

```
GET /api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/lv HTTP/1.1<CR><LN>
```

```
Host: {Host}<CR><LN>
```

```
User-Agent: {USER_AGENT}<CR><LN>
```

```
X-Auth-Token: {TOKEN}<CR><LN>
```

```
Content-Type: application/json<CR><LN><CR><LN>
```

Expected Response:

```
HTTP/1.1 200 OK<CR><LN>
```

```
Server: nginx<CR><LN>
```

```
Date: Tue, 04 Sep 2018 22:35:06 GMT<CR><LN>
```

```
Content-Type: application/json<CR><LN>
```

Transfer-Encoding: chunked<CR><LN>

Vary: Cookie<CR><LN>

Allow: GET, POST, HEAD, OPTIONS<CR><LN><CR><LN>

{PAYLOAD_LENGTH_IN_HEXADECIMAL}<CR><LN>

{LAST_VALUE}<CR><LN>

0<CR><LN>

In the *path* specified above, replace the {DEVICE_LABEL} and {VARIABLE_LABEL} keys with the unique labels of the Ubidots device and variable you desired to retrieve data from.

```
GET
https://industrial.api.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/lv
GET
https://things.ubidots.com/api/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/lv
```

Get the last value from a variable (Educational users)

```
curl -X GET -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" https://things.ubidots.com/api/v1.6/devices/my-device/my-variable/lv
```

Get the last value from a variable (Industrial users)

```
curl -X GET -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" https://industrial.api.ubidots.com/api/v1.6/devices/my-device/my-variable/lv
```

HTTP Examples

In this section, you will find different examples that show you how to handle data with Ubidots to start monitoring and controlling your IoT Devices.

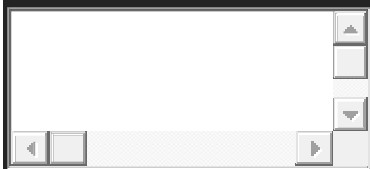
At the right side, you can find each example provided in different programming languages. Please, refer to your preferred programming language by clicking the tabs located on the right upper side of the page.

Sending a geopoint

Ubidots provides multiples ways to start tracking your IoT devices, so here we are going to show you how to send coordinates to Ubidots in order to start with a GPS IoT Solution.

To learn more about the Ubidots Maps Features, please check [this guide](#).

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"location":{"value": 1, "context":{"lat":37.773, "lng":-122.431}}}' https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device
```



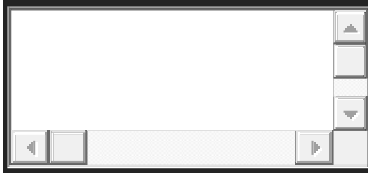
Sending a string value

The String values provide additional information to the dot which is being sent. As a basic example here, let's suppose that you are deploying a Weather Station which provides you the Weather Status as a String value. At this point, the Weather Status ought to be sent in the context of the dot.

To learn more of how to display the context of a dot in the Ubidots Dashboard, please check [this guide](#).

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"weather":{"value": 1, "context":{"weather-status":
```

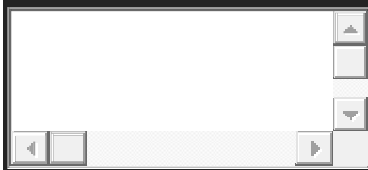
```
"sunny"}" https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device
```



Sending values in bulk from a datalogger

In case you desire to send information to Ubidots from a data logger which is in charge of taking samples for long periods of time, you can simply do it by managing the values registered plus the time when the value was registered.

```
curl -X POST -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -H "Content-Type: application/json" -d '{"my-sensor":[{"value": 27, "timestamp":1514808000000}, {"value": 30, "timestamp":1514808900000}, {"value": 31, "timestamp":1514809800000}, {"value": 29, "timestamp":1514810700000}, {"value": 27, "timestamp":1514768400000}]}' https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device
```



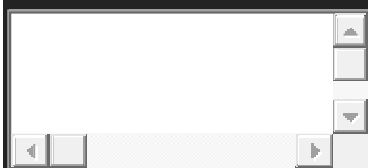
Retrieve the last value for control

This is typically used for remote control applications. Your devices would retrieve the last value of a variable, and then act based on that value. It is also used to remotely manage your devices, like for example changing a sampling rate or performing a remote reboot.

To learn more about the control features that Ubidots offer, please check [this guide](#).

Example Request

```
curl -X GET -H "X-Auth-Token: BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" "https://industrial.api.ubidots.com/api/v1.6/devices/my-new-device/control/lv"
```



Response codes

The Ubidots REST API uses the following response codes when you make an HTTP request, with them you can know your request status and errors if there are any:

Code	Meaning
200	Ok -- Successful request.
201	Created -- Successful request + an item (device or variable) was created.
202	Accepted -- The request has been accepted for processing, but the processing has not been completed.
204	One of the fields is incorrect and the request is not saved -- Please verify it's a valid JSON string and that the fields are the ones expected by the endpoint (string, object or float).
400	Bad Request -- Error due to an invalid body in your request. Please verify it's a valid JSON string and that the fields are the ones expected by the endpoint (string, object or float).
401	Invalid API key -- Please verify your API Key.
402	Payment required -- Please verify your balance.
403	Forbidden -- This token is not valid. Please verify your token.

Code	Meaning
404	Not Found -- We couldn't find the URL you're trying to access. This might be due to a wrong device label or ID, a wrong variable label or ID, or simply a typo in the URL of the request.
405	Method Not Allowed -- This API endpoint does not accept the method used. Check our API docs to see the allowed methods.
415	Unsupported media type -- The payload is in a format not supported by this method on the target resource.
420	You have exceeded your API limits -- To upgrade your API limits contacts the Ubidots team
423	Device does not receive data because it is disabled
429	Too many requests -- Many requests sent in a given amount of time ("rate limiting")
50*	Internal Error -- We're having issues with our servers. Please check our status page .

MQTT

MQTT is an **"Internet of Things"** connectivity protocol. It was designed as a n extremely lightweight **publish/subscribe** messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium"

(Source: MQTT.org).

MQTT is specially useful to push data from the cloud to your devices. Imagine a cloud-controlled device to open/close a door remotely. In the case of HTTP, the device would have to continuously make GET requests to the Ubidots server to see if there's a change in a variable, say "Door Control Variable", and then take an action depending on the last reading. This takes a lot of requests and it's not entirely a real-ti

me interaction since it depends of the polling frequency. With MQTT, the device can "listen" to the cloud and only get notified when there's a change in the variable. This way, the connection between the device and the cloud is left open but data only travels when is necessary, saving battery, network bandwidth and improving the real-time experience.

In this section you will find the documentation needed to handle your devices with Ubidots over MQTT.

For a detailed explanation of the **Ubidots MQTT API** Reference click [here](#)

Quality of Service

In MQTT, the quality of service, QoS for short, guarantees the delivery of a specific message. There are three levels of QoS:

- 0, At most once: This level does not guarantee the message delivery, it would be labeled as best-effort server QoS.
- 1, At least once: This level guarantees that the message is delivered at least one time to the receiver. The receiver answers with a puback packet, if this puback packet is not received the sender sends again the message.
- 2, Exactly once: Guarantees that the message is received only once by the receptor. It is the slowest packet interchange QoS as it requires two request/response flows.

Ubidots supports QoS up to 1 (at most once).

Broker URLs

API access can be over plain MQTT or secure MQTT with TLS, using the following endpoints based on your Ubidots Account type:

Non-TLS

Ubidots Account	Endpoint	Port
Educational	things.ubidots.com	1883

Ubidots Account	Endpoint	Port
Industrial	industrial.api.ubidots.com	1883

TLS

Ubidots Account	Endpoint	Port
Educational	things.ubidots.com	8883
Industrial	industrial.api.ubidots.com	8883

You can download the Ubidots PEM certificate for TLS [here](#)

Security Note: We strongly advise to use MQTT with TLS to make sure your data travels encrypted, avoiding the exposure of your API token and/or sensor data to third parties.

NOTE: To avoid future inconveniences, we advise to use both certificates in your device's auth routines. While the first one is the current valid certificate, we may change this after 2021.

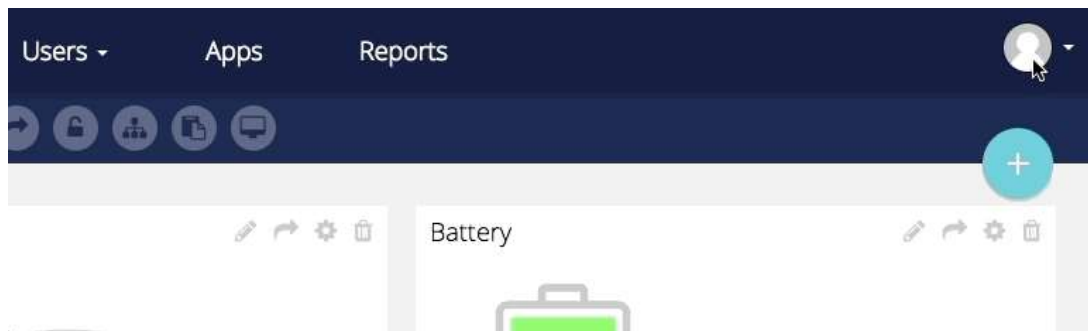
Fixed IPs

Ubidots also supports fixed IPs address for data ingestion, but we highly advise to our users to implement their firmware routines using the dns endpoints instead of fixed IPs, as these last ones may change in the future.

Ubidots Account	ipv4	ipv6	Port
Educational	50.23.124.68	2607:f0d0:2101:39::2	1883/8883
Industrial	169.55.61.243	2607:f0d0:3:b::2	1883/8883

MQTT Authentication

To interact with the Ubidots MQTT broker you'll need a token. A token is an unique key that authorizes your device to interact with Ubidots API.



To establish the connection with the MQTT broker you need to assign the username and password following the table below:

MQTT Credentials	Parameter
Username	Ubidots <i>token</i>
Password	any character - not required -

Send data (Publish)

To send data, set the unique Device label in the topic path.

Ubidots accepts MQTT data JavaScript Object Notation or JSON. JSON is a collection of name/value pairs in various programming languages, this is treated as an object, record, struct, dictionary, hash table, keyed list, or associative array. It is also human readable and language independent. An example of a JSON data type that Ubidots accepts can be referenced below:

```
{"temperature": {"value":10, "timestamp": 1534881387000, "context": {"machine": "1st floor"}}
```

The keys context and timestamp are optional.

NOTE 1: The Ubidots backend will take the latitude and longitude values inside any variable labeled as "gps", "location", position or "coordinates" automatically to get your device coordinates in all of the map widgets. You may change the default location variable from Ubidots web user interface.

NOTE 2: All timestamps must be in milliseconds. If you set a timestamp for a variable value, it will be stored with the date represented by it. You can easily convert dates to timestamps values [here](#).

If you wish to send more than one value, then you need to separate them using commas. Below you can reference a typical structure for MQTT payloads:

Variables	JSON structure
One	<code>{"VARIABLE-LABEL": {"value":VALUE, "timestamp":TIMESTAMP, "context":{"KEY":VALUE}}}</code>
Multiple	<code>{"VARIABLE-LABEL_1": {"value":VALUE, "context":{"KEY":VALUE}}, ... , "VARIABLE-LABEL_n": {"value":VALUE, "context":{"KEY": VALUE}}}</code>

keys context and timestamp are optional.

Examples

Note: All the bash examples from this section makes use of the Eclipse Mosquitto library, if you wish to test them please install it using the official instructions

- Single dot: `{"temperature": 27}`
- Multiple dots: `{"temperature": 27, "humidity": 55, ... , "pressure": 78}`

Including additional keys like context and timestamp

- Single dot: `{"position": {"value" : 1, "timestamp": 1514808000000, "context":{"lat":-6.2, "lng":75.4, "name" : "John"}}`
- Multiple dots:`{"temperature": {"value": 27, "timestamp": 1514808000000},`

"humidity": 55, "pressure": {"value": 78, "context":{"name" : "John"}}}

PUB

/v1.6/devices/{DEVICE_LABEL}/

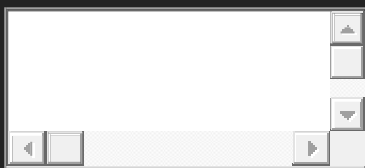
Send a single dot (Educational users)

```
mosquitto_pub -h "things.api.ubidots.com" -t "/v1.6/devices/weather-station" -u "p4uuT2O25AJwv7ncADLoVhQfImwRQW" -p 1883 -q 1 -m '{"temperature": 27}'
```



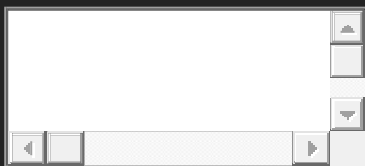
Send a single dot (Industrial users)

```
mosquitto_pub -h "industrial.api.ubidots.com" -t "/v1.6/devices/weather-station" -u "BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -p 1883 -q 1 -m '{"temperature": 27}'
```



Send a dot with coordinates

```
mosquitto_pub -h "industrial.api.ubidots.com" -t "/v1.6/devices/weather-station" -u "BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -p 1883 -q 1 -m '{"position": {"value": 1, "timestamp": 1514808000000, "context":{"lat":-6.2, "lng":75.4}}}'
```



Send multiple dots

```
mosquitto_pub -h "industrial.api.ubidots.com" -t "/v1.6/devices/weather-station" -u "BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -p 1883 -q 1 -m '{"temperature": {"value": 27, "timestamp": 1514808000000, "humidity": 55, "pressure": {"value": 78, "context":{"name" : "John"}}}'
```



Retrieve data (Subscribe)

Similar to the GET function in HTTP, subscribe is the method to obtain values with a huge difference: you do not have to be continuously ask the server for each value in your custom script. For example, if a variable's value changes, Ubidots will automatically update you (the user) of any changes. Thus saving data requests and processing time for your device and overall project functionality and expenses. Because of this, MQTT is a communication protocol advised to implement firmware routines to control actuators.

Retrieve the last value of a variable

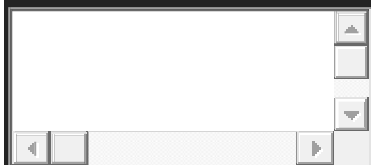
Replace the {DEVICE_LABEL} and {VARIABLE_LABEL} with the unique label of the device and the variable to subscribe for retrieving values update. If the specified Device or Variable do not exist, you will not able to establish the subscription with it.

SUB

```
/v1.6/devices/{DEVICE_LABEL}/{VARIABLE_LABEL}/lv
```

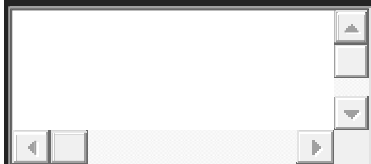
Subscribe to a variable (Educational users)

```
mosquitto_sub -h "things.ubidots.com" -t "/v1.6/devices/weather-station/temperature/lv" -u "p4uuT2O25AJwv7ncADLoVhQfImwRQW" -p 1883 -q 1
```



Subscribe to a variable (Industrial users)

```
mosquitto_sub -h "industrial.api.ubidots.com" -t "/v1.6/devices/weather-station/temperature/lv" -u "BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB" -p 1883 -q 1
```



MQTT Examples

In this section, you will find different examples that show you how use MQTT to implement several IoT applications.

At the right side, you can find each example provided in different programming languages. Please, refer to your preferred programming language by clicking the tabs located on the right upper side of the page.

GPS Tracker

Ubidots provides multiples ways to start tracking your IoT devices, so here we are going to show you how to send coordinates to Ubidots in order to start with a GPS IoT Solution.

To learn more about the Ubidots Maps Features, please check [this guide](#).

```
mosquitto_pub -p 1883 -h industrial.api.ubidots.com -t "/v1.6/devices/my-new-device" -u 'BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB' -m '{"location":{"value": 1, "context":{"lat":37.773, "lng":-122.431}}}' -d -q 1
```

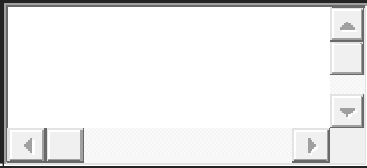


Weather description

The String values provide additional information to the dot which is being sent. As a basic example here, let's suppose that you are deploying a Weather Station which provides you the Weather Status as a String value. At this point, the Weather Status ought to be sent in the context of the dot.

To learn more of how to display the context of a dot in the Ubidots Dashboard, please check [this guide](#).

```
mosquitto_pub -p 1883 -h industrial.api.ubidots.com -t "/v1.6/devices/my-new-device" -u 'BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB' -m '{"weather":{"value": 1, "context":{"weather-status": "sunny"}}}' -d -q 1
```



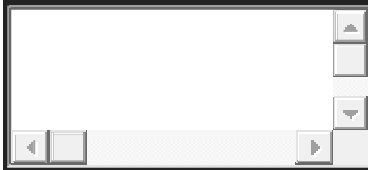
Retrieve the last value for control

This is the perfect solution for you in case you desire to control remotely the garage doors, home lights, alarm system, or any asset you desire.

To learn more about the control features that Ubidots offer, please check [this guide](#).

Example Request

```
mosquitto_sub -p 1883 -h industrial.api.ubidots.com -t "/v1.6/devices/my-new-device/control/lv" -u 'BBFF-Rfcgaxns6HIVb155WA0RhSY85xNDmB' -d -q 1
```



TCP / UDP

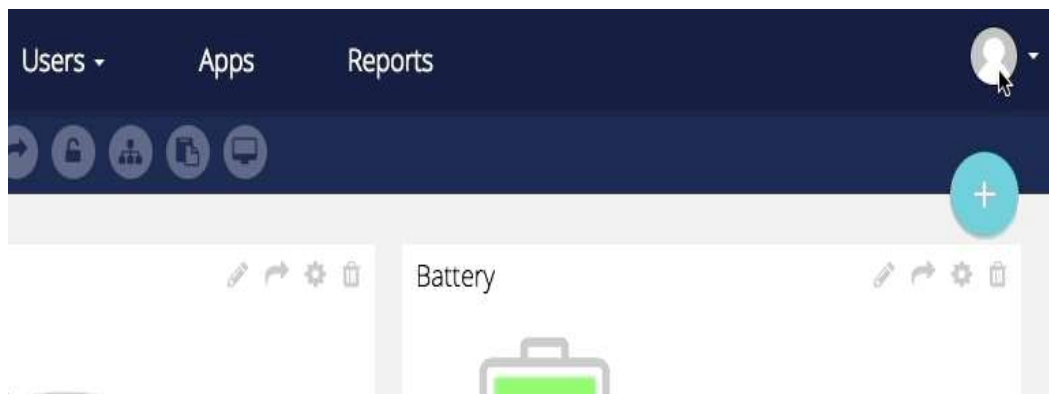
The Transmission Control Protocol, TCP, is one of the most popular data transport protocols, so much it is supported by default in most hardware devices. Its main advantage comes in the packet size, usually lower than packets sent using HTTP (which in fact, uses TCP as transport layer). TCP is "connection oriented", which means it implements confirmation methods to guarantee that data is received by the destinationary.

The User Datagram Protocol, UDP, is also a very popular protocol that, unlike TCP, it is "connectionless". This means it does not implements any confirmation method to guarantee that data is received by the destinationary. This can be an advantage in applications where simplicity and speed are more important than reliability.

TCP and UDP are similar in the sense that they share the structure of endpoint + port.

TCP Authentication

To send data to Ubidots using TCP or UDP broker you need your Ubidots TOKEN. The easiest way to get it is by clicking on "API Credentials" under your profile tab:



TCP URLs

API access can be over plain TCP or secure TCP with TLS, using the following endpoints based on your Ubidots Account type:

Non-TLS

Ubidots Account	Endpoint	Port
Educational	things.ubidots.com	9012
Industrial	industrial.api.ubidots.com	9012

TLS

Ubidots Account	Endpoint	Port
Educational	things.ubidots.com	Not Available
Industrial	industrial.api.ubidots.com	9812

You can download the Ubidots PEM certificate for TLS [here](#)

Security Note: We strongly advise to use TCP with TLS to make sure your data travels encrypted, avoiding the exposure of your API token and/or sensor data to third parties.

NOTE: To avoid future inconveniences, we advise to use both certificates in your device's auth routines. While the first one is the current valid certificate, we may change this after 2021.

Send Data

Structure

Ubidots created a pre-defined structure to understand the content of your TCP/UDP packets. If you wish to implement a custom structure, please get in touch with our support channel.

Single device:

```
{USER_AGENT}|POST|{TOKEN}|DEV_LABEL(:DEV_NAME)(@GLOBAL_TIMESTAMP)=>[VAR_LABEL:VALUE([$CONTEXT_KEY=CONTEXT_VAL]...)](@VARIABLE_TIMESTAMP)|end
```

Multiple devices:

```
{USER_AGENT}|POST|{TOKEN}|DEV_LABEL(:DEV_NAME)(@GLOBAL_TIMESTAMP)=>[VAR_LABEL:VALUE([$CONTEXT_KEY=CONTEXT_VAL]...)](@VARIABLE_TIMESTAMP); ...  
;DEV_LABEL(:DEV_NAME)(@GLOBAL_TIMESTAMP)=>[VAR_LABEL:VALUE([$CONTEXT_KEY=CONTEXT_VAL]...)](@VARIABLE_TIMESTAMP)|end
```

Where:

- {USER_AGENT}: Mandatory. Contains a characteristic string that allows to identify the application type, operating system, software vendor or software version of the requesting software user agent. Examples: ESP8266/1.0, Particle/1.2
- {DEVICE_LABEL}: Mandatory. The device label to publish values. If it does not exist, Ubidots will create it. Blank spaces are not allowed. Examples: Weather-station, living-room
- {DEVICE_Name}: Optional. The device name to publish values. If not set, the device name will be set using the device label. This parameter is useful when you have as device label your device MAC and you wish to show to final

users a friendly name. Blank spaces are not allowed.

Examples: Street-42-weather-station, living-room-floor-1

- {GLOBAL_TIMESTAMP}: Optional. Unix time stamp. If set, any variable value without timestamp will use this one to store values. Must be in milliseconds. If not set, Ubidots will add it automatically with the actual timestamp in UTC.

Examples: 1534282544356

- {VARIABLE_LABEL}: Mandatory. The variable label that will store the dot. If it does not exist, Ubidots will create it. You may send multiple variables splitting them by commas.

Examples: temperature, wind-speed

- {VALUE}: Mandatory. Dot value. You can use a dot to set float numbers.

Examples: 1.0005, 2

- {CONTEXT_KEY}: Optional. If set, you must specify a context value. You can add as many context key as you wish splitting them with a '\$' char.

Examples: lat, lng, name, surname.

- {CONTEXT_VALUE}: Optional. The context key value.

Examples (based on keys above): -6.2, 75.4, "John", "Smith"

- {VARIABLE_TIMESTAMP}: Optional. Unix time stamp. If set, the variable will use this one to store dots. This parameter has precedence among the global timestamp. Must be in milliseconds. If not set, Ubidots will add it automatically with the actual timestamp in UTC.

Examples: 1534282544356

Server Response

[OK|...]

ERROR|400

- Valid messages: Ubidots will answer Ok if the request is correct. If multiple values are sent, ok|OK... strings (one per variable in the request) splitted by pipelines '|' will be returned.
- Non valid messages: Ubidots will answer ERROR|400 if the message is not valid.

Stream Examples

- Variable with multiple context values:

```
ubidots/1.0|POST|{TOKEN}|weather-station=>temperature:20$day-hour=11$day-minute=23|end
```

- Variable with same value to multiple devices:

```
ubidots/1.0|POST|{TOKEN}|weather-station-denver=>temperature:20;weather-station-london=>temperature:20|end
```

Examples

Send a single dot to a device with name *green-truck* and device label *5b7356ccbdbdbd594df54555* stored inside the variable *speed*:

```
ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQfImwRQW|5b7356ccbdbdbd594df54555:green-truck=>speed:2|end
```

Send a single dot to a device named as *green-truck* with device label *5b7356ccbdbdbd594df54555* stored inside the variable *speed* at 2017/12/31 8:00 UTC:

```
ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQfImwRQW|5b7356ccbdbdbd594df54555:green-truck=>speed:2@1514707200000|end
```

Send a single dot to a device labeled as *green-truck* with api label *5b7356ccbdbdbd594df54555* stored inside the variable *speed* at 2017/12/31 8:00 UTC with latitude, longitude and other keys inside its context:

```
ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQfImwRQW|5b7356ccbdbdbd594df54555:green-truck=>speed:2$lat=-6.2$lng=75.4$name=John$surname=Smith@1514707200000|end
```

Send multiple values

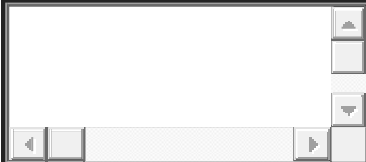
```
ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQfImwRQW|weather-station=>temperature:20,humidity:35|end
```

Send values to multiple devices

ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQflmwRQW|weather-station-toronto=>temperature:20,humidity:35;weather-station-washington=>temperature:20,humidity:35|end

Send a single dot (Educational users)

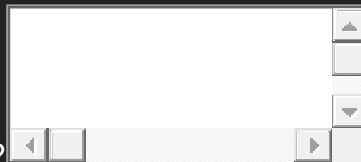
```
echo -n "ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQflmwRQW|5b7356ccbdbd594df54555:green-truck=>speed:2|end" | telnet things.ubidots.com 9012
```



Send a single dot (Industrial users)

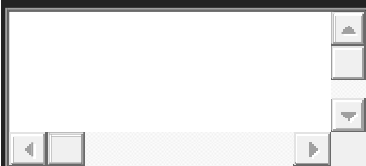
```
echo -n "ESP8266/1.0|POST|BBFF-kiFdJvtV3OWTY56M0rKluUGm71lzzn6kboWpbWpfhJFah7fL6p1kcks|green-truck:5b7356ccbdbd594df54555=>speed:2|end" | telnet
```

```
et industrial.api.ubidots.com 9012
```



Send multiple dots (Educational users)

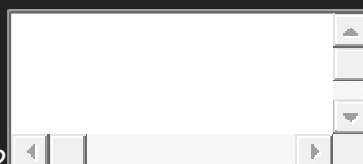
```
echo -n "ESP8266/1.0|POST|p4uuT2O25AJwv7ncADLoVhQflmwRQW|weather-station=>temperature:20,humidity:35|end" | telnet things.ubidots.com 9012
```



Send multiple dots (Industrial users)

```
echo -n "ESP8266/1.0|POST|BBFF-kiFdJvtV3OWTY56M0rKluUGm71lzzn6kboWpbWpfhJFah7fL6p1kcks|weather-station=>temperature:20,humidity:35|end" | telnet indu
```

```
strial.api.ubidots.com 9012
```



[Retrieve Data](#)

Structure

Ubidots TCP/UDP supports the payload request structure below for retrieving dots values

```
{USER_AGENT}|LV|{TOKEN}|{DEVICE_LABEL}:{VARIABLE_LABEL}|end  
{USER_AGENT}|GET|{TOKEN}|{VARIABLE_ID}|end
```

where:

- {USER_AGENT}: *Mandatory*. Contains a characteristic string that allows to identify the application type, operating system, software vendor or software version of the requesting software user agent. Examples: ESP8266/1.0, Particle/1.2
- {DEVICE_LABEL}: *Mandatory*. The device label to publish values. If it does not exist, Ubidots will create it. Blank spaces are not allowed. Examples: Weather-station, living-room
- {VARIABLE_LABEL}: *Mandatory*. The variable label that stores the dot. Examples: temperature, wind-speed
- {VARIABLE_ID}: *Mandatory*. The variable unique identifier that stores the dot. Examples: 573dc1ee76254241c57a4616

Server Response

OK|Number

ERROR|400 , ERROR|404

- Valid message: Ubidots will answer Ok| followed by the dot value if the request is correct.
- Non valid message: Ubidots will answer ERROR|400 if the message is not valid or ERROR|400 if the message contains a variable or device that does not exist.

Examples

Get the last value from the variable *speed* inside a device labeled with api label *green-truck*:

```
ESP8266/1.0|LV|p4uuT2O25AJwv7ncADLoVhQfImwRQW|green-truck:speed|end
```

Get the last value from the variable *speed* inside a device labeled with api label *green-truck* using the *speed* variable unique id:

```
ESP8266/1.0|GET|p4uuT2O25AJww7ncADLoVhQflmwRQW|573dc1ee76254241c57a4616|end
```

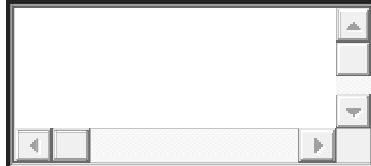
Get last variable value (Educational users)

```
echo -n "ESP8266/1.0|LV|p4uuT2O25AJww7ncADLoVhQflmwRQW|green-truck:speed|end" | telnet things.ubidots.com 9012
```



Get last variable value (Industrial users)

```
echo -n "ESP8266/1.0|LV|BBFF-kiFdJvtV3OWTY56M0rKluUGm71lzzn6kboWpbWpfhJFah7fL6p1kcks|green-truck:speed|end" | telnet industrial.api.ubidots.com 9012
```



[Utils](#)

Ubidots also supports to retrieve the actual UTC timestamp from our servers.

RESULT:

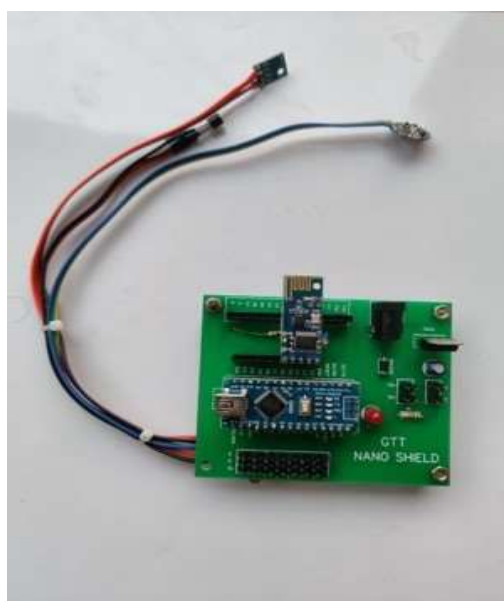


Fig no 15: Transmitter

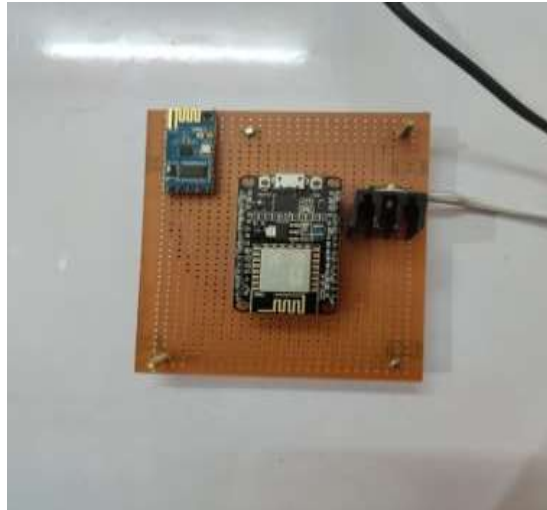


Fig no 16:Receiver

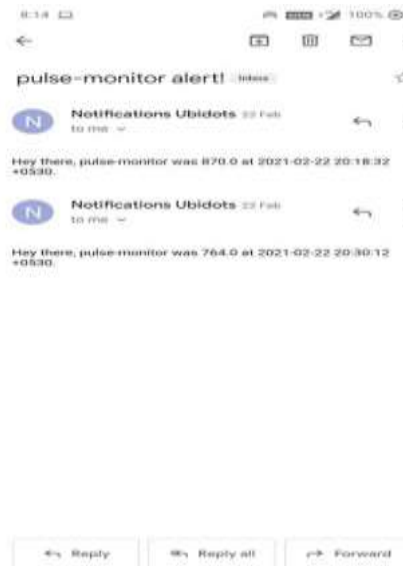


Fig no 17:Notification Alert

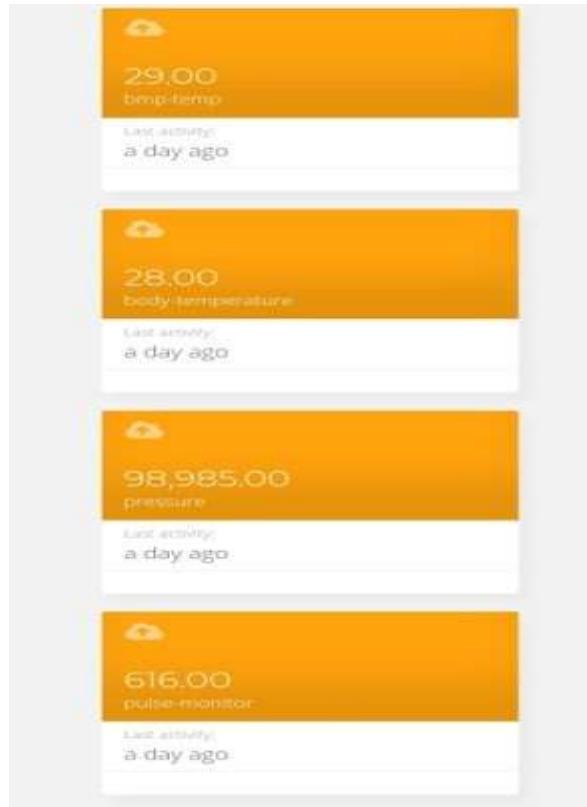


Fig No 18: atmospheric Measurements

CONCLUSION:

We can state that with the existing technology it is difficult to identify, track and locate a fishing boat or any domestic vessel offshore by the means of RADAR / Satellitebased optimized search for vessels near victim vicinity, or by sending distress signals such as Emergency Positioning Indicating Radio Beacon (EPIRB), Search and Research Transponder (SART) since these methods are proven to be costly and hence it is nearly impossible for the government to completely monitor the coast line and track each and every position of the fishermen going into the sea. With our proposed model, we can track the location of the boats up to 10 - 15 miles from off coastline by simply tracking the GPS location of their smart phones which can further be extended by using signal repeaters, almost doubling the range. This method is feasible and cost effective because of the abundant use of smart phones with GPSbased location services enabled in them. With our system we can ensure to easily track the number of people entering the sea and coming back to the shore with the date and time log of entering and exiting the harbour, hence it is easy to identify the

missing native personnel's or illegal immigrant's activities. The entire security is based on User's Private Pin which is unique to each user, once authorized it generates a personal QR code on the server which activates or ends the trip only when scanned using the application with the registered handset.

REFERENCES

- [1] C. Sheeba thangapushpam “Intelligent boundary alert system using GPS” IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, pISSN: 2278-8727 PP 19-23
- [2] C.Vinothkumar and B.Arunkumaran “Implementation of GPS Based Security System for Safe Navigation of Fisherman Autoboat” International Journal for Research and Development in Engineering (IJRDE) ISSN: 2279-0500 Special Issue: pp- 465-469
- [3] S. Kiruthika and N. Rajasekaran “Implementation Of GPS Based Surveillance Navigation System For Fisherman” International Journal of Scientific Research and Engineering Studies (IJSRES) Volume 1 Issue 5, November 2014 ISSN: 2349-8862 Page 58
- [4] M.Sivaramaganesh, M.Ramya, V.Gowtham, T.Bharathi, G.Jeevitha “Implementation Of Maritime Border Alert System” International Journal Of Innovative Research In Electrical, Electronics, Instrumentation and Control Engineering Vol. 2, Issue 3, March 2014
- [5] Dhinakaran, K., Srinath, S., Sriram, S., and Venkateshwar, R. (2017). “Gps based tracking system for transit objects.” Science Technology Engineering &Management (ICONSTEM), 2017 Third International Conference on, IEEE. 194–201.
- [6] Hofmann-Wellenhof, B., Lichtenegger, H., and Collins, J. (2012). Global positioning system: theory and practice. Springer Science & Business Media.
- [7] Hu, P., Guo, Q., Yuan, C., and Zhang, X. (2011). “Design and implementation of 1.6 ghzepirb antenna.” Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC), 2011, Vol. 1, IEEE. 429–432.
- [8] Prabakaran, K., Kumar, N. R., Punitha, G., Asmi, S., and Paulchamy, B. (2013). “A noval approach of geofencing and geotagging system based sea border identification using embedded system.” Current Trends in Engineering and Technology (ICCTET), 2013 International Conference on, IEEE. 338–341.
- [9] Reclus, F. and Drouard, K. (2009). “Geofencing for fleet & freight management.”Intelligent Transport Systems Telecommunications, (ITST), 2009 9th International Conference on, IEEE. 353–356
- [10] Shah, A. T. and Parihar, V. R. (2017). “Overview and an approach for qr-code based messaging and file sharing on android platform in view of security.”

Computing Methodologies and Communication (ICCMC), 2017 International Conference on, IEEE. 371–373