**SMART E UNIFORM FOR SOLDIERS USING IOT**

**ABSTRACT**

Soldiers, who courageously defend our nation, face diverse weather conditions during their service. To safeguard them, a solar-powered climate-adjustable E-uniform has been developed. This innovative uniform dynamically adjusts to provide warmth or cooling, depending on the prevailing weather conditions. The uniform operates in two modes: summer and winter, allowing soldiers to regulate their body temperature as needed. Key components of the uniform include a temperature sensor, a microcontroller, a power supply, and heating/cooling mechanisms. The temperature sensor continuously monitors the soldier's body temperature, while the microcontroller analyzes this data to activate the appropriate heating or cooling devices as required.

**INTRODUCTION**

Soldiers, the backbone of our nation's defense, endure various challenges, especially adverse weather conditions, while safeguarding our country. Our project aims to address these challenges by introducing a revolutionary solution: the Solar Guard Gear. This innovative uniform is designed to offer unparalleled safety and security in any weather scenario, providing soldiers with versatility, comfort, and lightness. The Solar Guard Gear integrates flexible solar panels into its fabric, enabling soldiers to harness and store solar energy throughout the day. This stored energy powers essential components of the uniform, such as communication devices, GPS systems, and sensors. By reducing reliance on conventional batteries, which are heavy and require frequent replacement, the Solar Guard Gear enhances soldiers' mobility and endurance on the battle field. In addition to sustainability, the solar Guard Gear offers numerous advantages. By eliminating the need for traditional batteries, it reduces the overall weight of the uniform, enhancing soldiers' agility. Moreover, the ability to generate power on-site decreases reliance on supply networks, mitigating the risk of power shortages during critical missions., including the need for robust and flexible solar panels capable of withstanding combat conditions, as well as efficient power management systems to optimize energy usage. Nevertheless, we believe that ongoing research and innovation will overcome these hurdles, making Solar Guard Gear an indispensable asset in military technology. Our project employs advanced components, including Peltier modules, microcontrollers, and temperature sensors, to ensure optimal functionality. The Arduino Uno regulates the uniform's components based on user commands, while the Peltier module serves as a temperature regulator. This sophisticated system enables the Solar Guard Gear to provide unparalleled safety and protection from extreme weather conditions.

**LITERATURE SURVEY:**

|  |  |  |
| --- | --- | --- |
| **TITLE** | **AUTHOR** | **CONTRIBUTION** |
| IoT-based Smart Wearable Jacket for Soldier Security | P. Kumar, M. Singh,2020 | Proposed a wearable system to monitor heart rate, temperature, and location using IoT and GSM for real-time soldier tracking. |
| IoT Enabled Smart Jacket for Army Applications | R. Sharma, S. Rajput,2021 | Introduced a jacket embedded with health sensors and GPS module that sends alerts using GSM. Integrated a panic button for emergency signaling. |
| Smart Military Uniform Using Wearable Electronics | K. Bansal et al.2019 | Developed a prototype of a smart uniform using wearable sensors for health, location, and environmental conditions with LoRa-based communication. |
| An IoT-based System for Soldier Monitoring | A. Joshi, T. Verma,2022 | Designed a wearable with ESP32, GPS, heart rate, and temperature sensors to transmit data to a cloud dashboard and alert using Twilio SMS. |

**EXISTING SYSTEM:**

This system monitors various environmental and physiological parameters for soldiers deployed in challenging conditions. It utilizes wireless modules, diverse sensors and a microcontroller. By integrating the soldier's body temperature and external weather conditions, the system adjusts the uniform's temperature accordingly. Additionally, it can transmit data to a remote server for analysis and alerts. Solar panels embedded in the uniform's fabric collect solar energy to power the system.

**DISADVANTAGE:**

1. **Depends on sunlight** – Solar panels may not work well in cloudy or dark environments.
2. **Can be bulky** – Adding electronics to the uniform may make it heavy or uncomfortable.
3. **Heating might not be even** – Peltier elements may not heat or cool all parts of the body equally.
4. **Wireless signals may fail** – Communication may not work in remote or signal-blocked areas.
5. **Expensive to make** – Advanced sensors and materials can increase the overall cost.
6. **Not fully waterproof** – Electronics may get damaged in rain or harsh weather.

**PROPOSED SYSTEM**

The proposed system is a **smart embedded wearable module** designed for soldiers operating in extreme and hazardous environments. It integrates **environmental monitoring**, **personal safety control**, and **automated thermal regulation** using a combination of sensors and actuators powered by **ESP8266** and controlled remotely via **IoT connectivity**.

**1. Toxic Gas Detection**

* The **Gas sensor** continuously monitors for the presence of harmful gases (e.g., CO, methane, LPG).
* When toxic gas is detected beyond a safe threshold, the system:
  + Activates a **cooling fan** for ventilation.
  + Sends an **alert to a remote server** via Wi-Fi using ESP8266.

**2. Proximity & Obstacle Monitoring**

* An **Ultrasonic sensor** is embedded to detect nearby objects or moving threats.
* Useful for:
  + Detecting environmental obstructions or potential hazards.
  + Enhancing safety during operations in low visibility or confined areas.

**3. Dynamic Body Temperature Regulation**

* A **Peltier module** is used to heat or cool the soldier’s uniform based on surrounding temperature conditions.
* Controlled by the **ESP8266** via a **relay module**.
* Automatically activates when ambient conditions exceed comfort thresholds.

**4. Smart Cooling with Fan Control**

* A **DC fan** assists in cooling during overheating scenarios, activated based on:
  + High ambient temperature.
  + High body temperature.
  + Harmful gas presence.

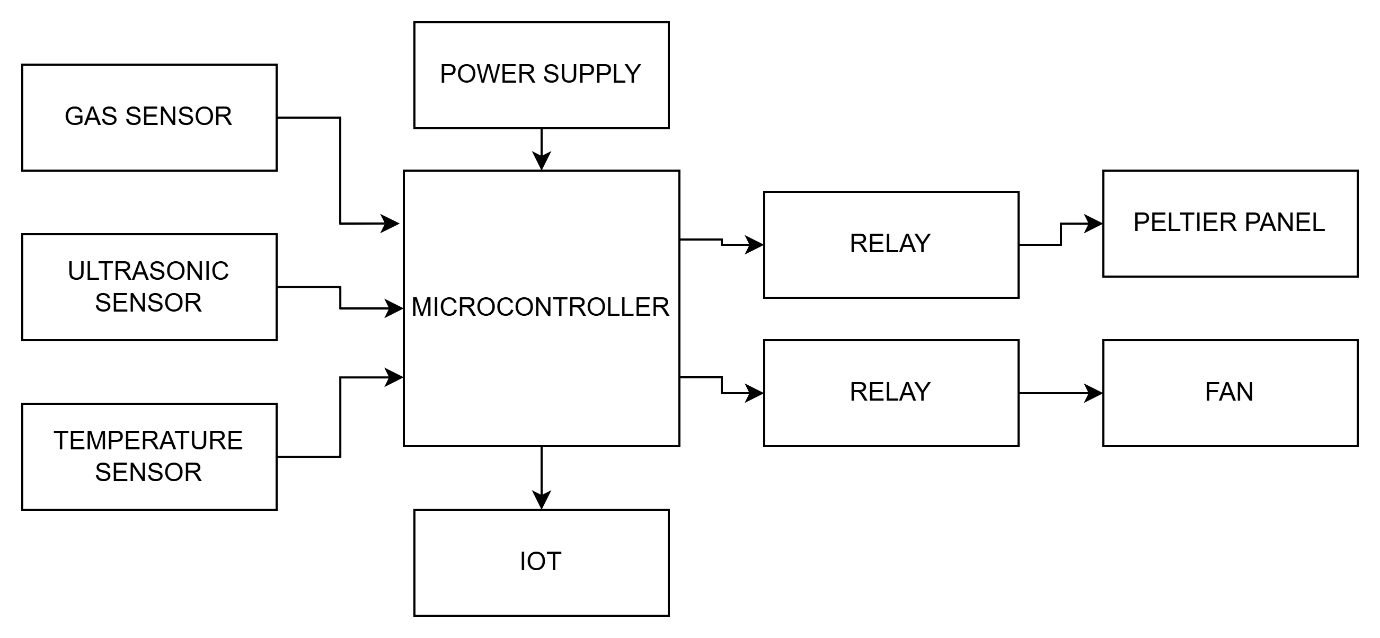
**5. IoT Connectivity (ESP8266)**

* Sends real-time data (temperature, gas levels, obstacle distance) to a **cloud dashboard**.
* Enables **remote monitoring** by commanders or control units.
* Can be integrated with platforms like , **ThingSpeak**, or **mqtt.**.

**ADVANTAGES:**

* **Enhanced Soldier Safety**
  + Detects toxic gases early and triggers automatic ventilation, reducing health risks in dangerous environments.
* **Real-Time Monitoring**
  + Sends live data (gas levels, temperature, obstacles) to a remote server for continuous situational awareness.
* **Smart Temperature Control**
  + Automatically regulates body temperature using a Peltier module and fan to ensure soldier comfort in extreme weather.
* **Obstacle & Proximity Detection**
  + Ultrasonic sensor improves awareness of nearby threats or obstacles, especially in low-visibility conditions.
* **IoT Integration**
  + Allows commanders to monitor soldiers' environment and health remotely via cloud platforms like Blynk or ThingSpeak.
* **Energy-Efficient Operation**
  + Components like fans and Peltier modules are activated only when needed, saving energy and improving battery life.
* **Multi-Functional System**
  + Combines health monitoring, environmental sensing, and smart control in one compact system.
* **Improved Operational Readiness**
  + Alerts and automation reduce manual checks and help soldiers stay focused on their mission.

**BLOCK DIAGRAM**

****

**HARDWARE REQUIREMENTS**

Microcontroller

Ultrasonic sensor

Gas sensor

Peltier panel

Fan

Relay

DHT11 sensor

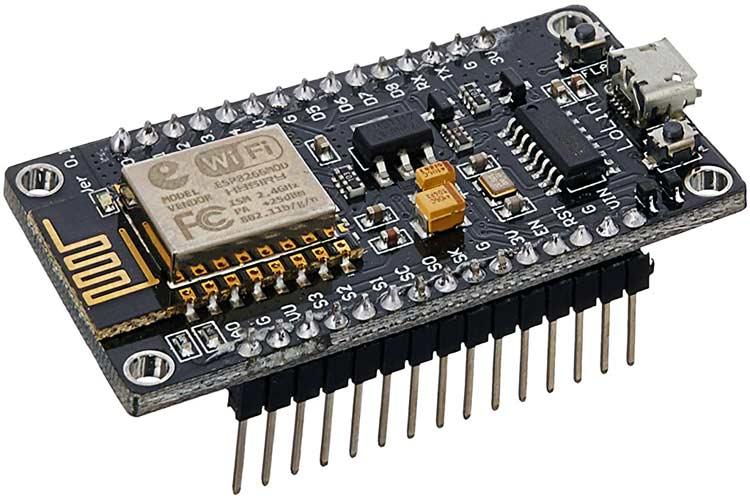
**SOFTWARE REQUIREMENTS**

Arduino ide

Embedded C

**HARDWARE REQUIREMENTS:**

**NODE MCU**

****

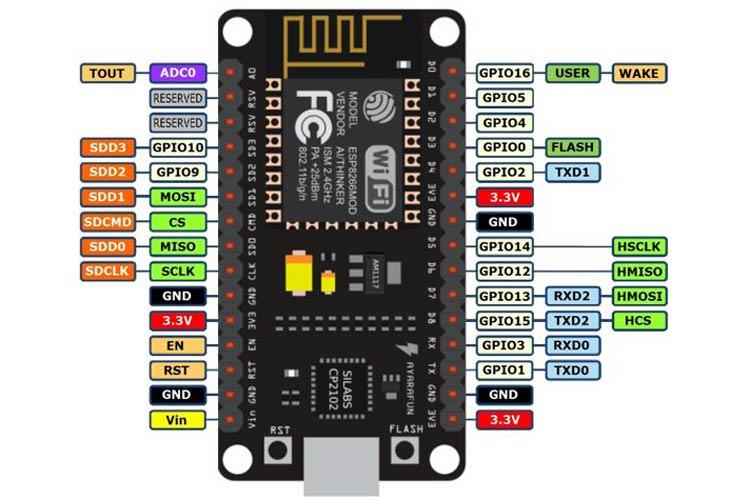
NodeMCU is an open-source Lua based firmware and **development board** specially targeted for IoT based Applications. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.

NodeMCU Development Board Pinout Configuration

|  |  |  |
| --- | --- | --- |
| **Pin Category** | **Name** | **Description** |
| Power | Micro-USB, 3.3V, GND, Vin | **Micro-USB:** NodeMCU can be powered through the USB port  **3.3V:** Regulated 3.3V can be supplied to this pin to power the board  **GND:** Ground pins  **Vin:**External Power Supply |
| Control Pins | **EN, RST** | The pin and the button resets the microcontroller |
| Analog Pin | A0 | Used to measure analog voltage in the range of 0-3.3V |
| GPIO Pins | GPIO1 to GPIO16 | NodeMCU has 16 general purpose input-output pins on its board |
| SPI Pins | SD1, CMD, SD0, CLK | NodeMCU has four pins available for SPI communication. |
| UART Pins | TXD0, RXD0, TXD2, RXD2 | NodeMCU has two UART interfaces, UART0 (RXD0 & TXD0) and UART1 (RXD1 & TXD1). UART1 is used to upload the firmware/program. |
| I2C Pins |  | NodeMCU has I2C functionality support but due to the internal functionality of these pins, you have to find which pin is I2C. |

NodeMCU ESP8266 Specifications & Features

* Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
* Operating Voltage: 3.3V
* Input Voltage: 7-12V
* Digital I/O Pins (DIO): 16
* Analog Input Pins (ADC): 1
* UARTs: 1
* SPIs: 1
* I2Cs: 1
* Flash Memory: 4 MB
* SRAM: 64 KB
* Clock Speed: 80 MHz
* USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
* PCB Antenna
* Small Sized module to fit smartly inside your IoT project

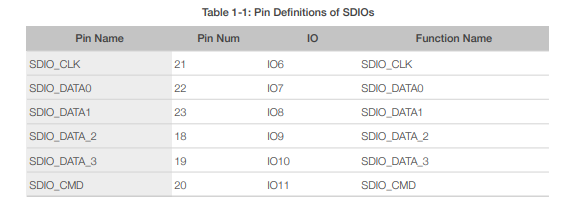


**General Purpose Input/Output Interface (GPIO)**

ESP8266EX has 17 GPIO pins which can be assigned to various functions by programming the appropriate registers. Each GPIO can be configured with internal pull-up or pull-down, or set to high impedance, and when configured as an input, the data are stored in software registers; the input can also be set to edge-trigger or level trigger CPU interrupts. In short, the IO pads are bidirectional, non-inverting and tristate, which includes input and output buffer with tristate control inputs. These pins can be multiplexed with other functions such as I2C, I2S, UART, PWM, IR Remote Control, etc.

**Secure Digital Input/Output Interface (SDIO)**

ESP8266EX has one Slave SDIO, the definitions of which are described below. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.



**Serial Peripheral Interface (SPI/HSPI)**

ESP8266EX has 3 SPIs.

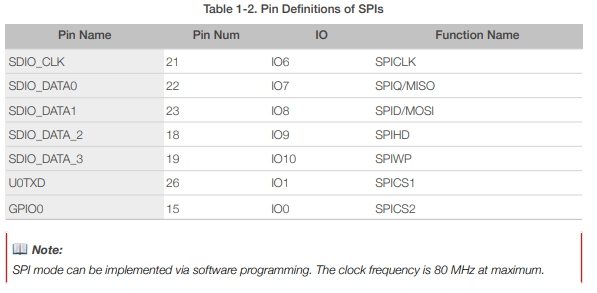
One general Slave/Master SPI

One Slave SDIO/SPI

One general Slave/Master HSPI

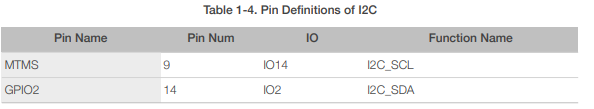
Functions of all these pins can be implemented via hardware. The pin definitions are described as below.

**General SPI (Master/Slave)**



I2C Interface

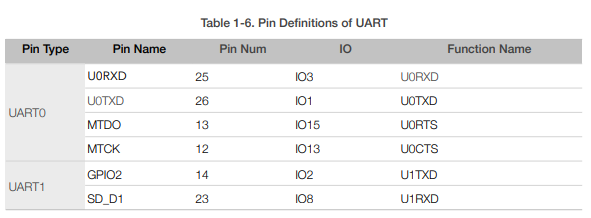
ESP8266EX has one I2C used to connect with micro-controller and other peripheral equipments such as sensors. The pin definition of I2C is as below.



Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized via software programming, the clock frequency reaches 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

**Universal Asynchronous Receiver Transmitter (UART)**

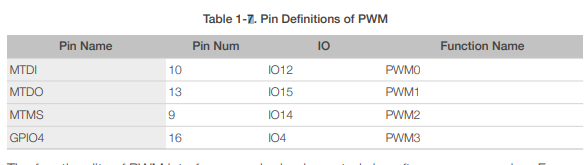
ESP8266EX has two UART interfaces UART0 and UART, the definitions are as below



Data transfers to/from UART interfaces can be implemented via hardware. The data transmission speed via UART interfaces reaches 115200 x 40 (4.5 Mbps). UART0 can be used for communication. It supports fluid control. Since UART1 features only data transmit signal (Tx), it is usually used for printing log.

**Pulse-Width Modulation (PWM)**

ESP8266EX has four PWM output interfaces. They can be extended by users themselves. The pin definitions of the PWM interfaces are defined as below.



The functionality of PWM interfaces can be implemented via software programming. For example, in the LED smart light demo, the function of PWM is realized by interruption of the timer, the minimum resolution reaches as much as 44 ns. PWM frequency range is adjustable from 1000 μs to 10000 μs, i.e., between 100Hz and 1 kHz. When the PWM frequency is 1 kHz, the duty ratio will be 1/22727, and over 14 bit resolution will be achieved at 1 kHz refresh rate.

**Functional Overview**

This protocol uses the SDIO mode of the ESP8266 to communicate with other processor's SPI hosts. The electrical interface is connected through signal line No.4, including the SCLK, MOSI, MISO and interrupt signal No.1 in the SPI protocol (note: no CS signal). Downloading the ESP8266 SDIO can be different from downloading other programs. When the ESP8266 starts, the system reads the pin shared by the SPI interface and the SDIO interface by default. Therefore, the SDIO module communication protocol should be used. The ESP8266 should start in the SDIO mode, and then, the host will start the chip in the ESP8266 RAM through the SDIO downloaded programs. The majority of the programs that directly use CPU CACHE to call FLASH can be burnt to the FLASH chip connected to the HSPI interface beforehand. Data received or sent by the ESP8266 SDIO is processed directly by the DMA module that supports linked list index. The ESP8266 can receive and send the SDIO packets efficiently without using the CPU. It does so through the address of the memory map linked list.

and re-transfer sketch, prior to interfacing outside voltage to AREF

**SOFTWARE TIPS**

While boot loading Atmega8 chip with Arduino 0010, there is an order (- i800) that makes bootloader defer 10 minutes. Thus, in the event that you need to utilize bootloader, use order line rather than IDE, eliminating "– i800" order and adding "– F" order, or use Arduino 0007 IDE. To transfer draws Arduino 0010 turns out great.

**ARDUINO S3v3 NEW FEATURES**

* full viable with Shield Boards (Version 2 is the main Arduino Board not viable with Shield Boards as a result of ICSP header wrong position, and tall parts);
* A Vcc LP channel to lessen commotion level on ADC;
* auto reset include;
* auto reset empower/impair jumper, to dodge not wanted resetting;
* arduinoDiecimila viable reset pin;
* pin13 locally available drove, with current limiter resistor;
* TX and RX locally available leds;
* power drove with suitable current limiter resistor (less 20mA of consumption);
* jumper to impair sequential correspondence and to empower RX outer draw down resistor, to evade "RX skimming blunder". This element permits to utilize computerized pin0 and pin1 as an ordinary pin, when sequential correspondence isn't required;
* all comparative segments (diodes, semiconductors, leds, capacitors) has a similar board direction (to commits simpler to mount with less errors);
* no wires between cushions, more space between wires, bigger wires, bigger cushions (better for drawing, binding and penetrating, with no shortcircuits, patching extensions or open wires in erosion);
* just 3 wire spans;
* electrolitic capacitor (in sequential to TTL circuit) changed to bipolar sort (to keep away from rearranged voltage issue when sequential link isn't associated);

All jumpers are correct point type, to permit Shield Boards use.

**ULTRASONIC SENSOR:**

Ultrasonic sensors are electronic devices that determine a target's distance. They work by emitting ultrasonic sound waves and converting those waves into electrical signals. Furthermore, ultrasonic travel at a faster rate than audible sounds. Therefore, ultrasonic sensor work involves sound waves to find the distance to an item. A transducer is also there to transmit and receive ultrasonic pulses. These pulses help to communicate information about an object within range.  Further, this detail can be applied in various applications including industrial. Let’s discuss in detail how ultrasonic sensor works.

**Working of an Ultrasonic Sensor**

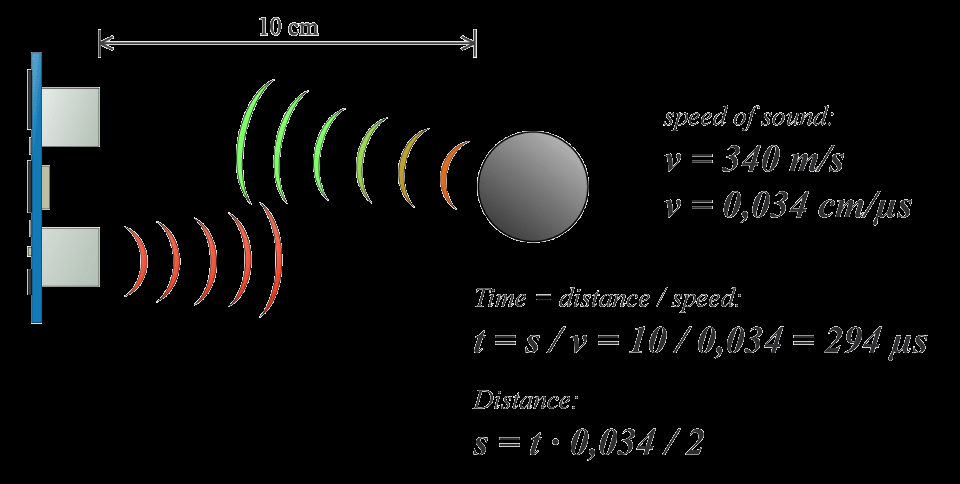
Ultrasonic sensors operate by emitting sound waves at frequencies that are too high for humans to hear. The sensor's transducer serves as a microphone to receive and transmit ultrasonic sound. They also use a single transducer to send and receive pulses. Further, the sensor measures the total time taken to deliver and receive an ultrasonic pulse and calculates the target's distance.



**Ultrasonic Sensor Working Principle**

The working principle of an ultrasonic sensor revolves around the emission and reception of high-frequency sound waves. These waves range from 20 kHz which is beyond the range of human hearing. Here's a simplified explanation of ultrasonic sensor operation:

* **Emitting Sound Waves:** The ultrasonic sensor generates a burst of ultrasonic sound waves, usually in the range of 20 kHz to 65 kHz. These sound waves travel through the air towards the target object.
* **Bouncing Off Objects:** When the sound waves encounter an object in their path, they bounce off the surface of the object.
* **Measuring the Return Time:** The sensor measures the time it takes for the emitted sound waves to bounce back after hitting the object. This time interval is extremely short, typically measured in microseconds.
* **Calculating Distance:** The sensor figures out how far an object is by using the speed of sound in the air. It does this by multiplying the time it takes for sound waves to go to the object and then coming back by the speed of sound.
* **Output Data:** Last, the sensor provides this distance information as an output, which can be used in various applications, such as obstacle detection, object positioning, or navigation.

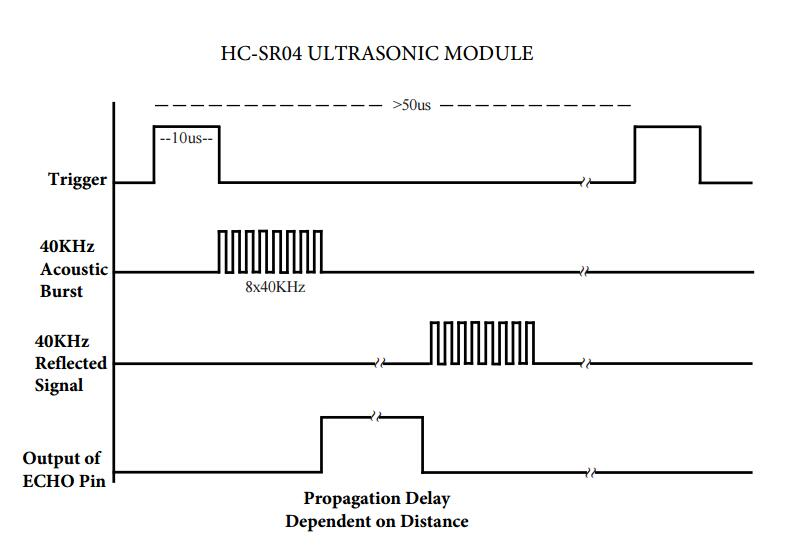


The above principle determines the ultrasonic sensor working. Now, let’s discuss the application areas of these [**sensors**](https://www.theiotacademy.co/blog/sensor).

Ultrasonic Sensor Application

Ultrasonic sensors are used in many areas of engineering. In automation, robotics, and instrumentation, "no-contact" distance measuring is highly helpful. Below are the common uses and functions of ultrasonic sensors:

* **Distance Measurement:** Ultrasonic sensors are commonly used to measure distances between the sensor and an object. By emitting sound waves and measuring the time it takes for them to bounce back, the sensor can calculate the distance accurately.
* **Object Detection:** In robotics and autonomous systems, ultrasonic sensors are crucial for detecting obstacles in the environment. It can determine whether an object is present within a specific range.
* **Positioning and Navigation:** Ultrasonic sensors are employed in robotics and autonomous vehicles for accurate positioning and navigation. By continuously measuring distances to surrounding objects, these systems can map their environment and make real-time decisions.
* **Tank Level Monitoring:** Ultrasonic sensor working seen in industries like agriculture. It is used to monitor the levels of liquids in large tanks or silos.
* **Collision Avoidance:** In automotive safety systems, ultrasonic sensors assist in collision avoidance by detecting nearby objects and providing warnings or assisting with parking.

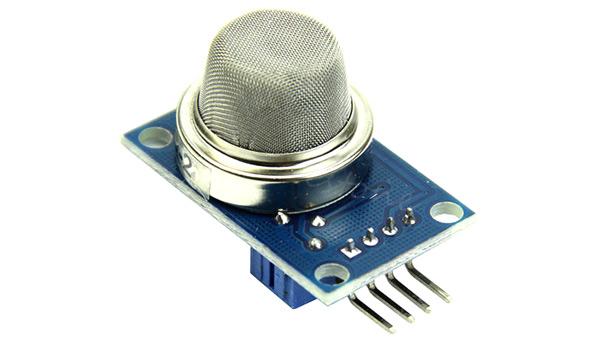


**Other Common** **Applications of Ultrasonic Sensors Include:**

* Ultrasonic sensors can measure the coil/roll's diameter.
* Useful in quality control.
* Applicable in the Detection of proximity.
* Helpful to position robotic arms through robotic sensing.
* Used in security system management.

**GAS SENSOR:**

A gas sensor is a device which detects the presence or concentration of gases in the atmosphere. Based on the concentration of the gas the sensor produces a corresponding potential difference by changing the resistance of the material inside the sensor, which can be measured as output voltage. Based on this voltage value the type and concentration of the gas can be estimated.



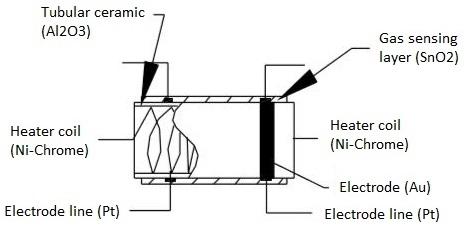
The type of gas the sensor could detect depends on the sensing material present inside the sensor. Normally these sensors are available as modules with comparators as shown above. These comparators can be set for a particular threshold value of gas concentration. When the concentration of the gas exceeds this threshold the digital pin goes high. The analog pin can be used to measure the concentration of the gas.

**Construction**

All Of the above-listed types, the most commonly used gas sensor is the Metal oxide semiconductor based gas sensor. All Gas sensors will consist of a sensing element which comprises of the following parts.

1. Gas sensing layer
2. Heater Coil
3. Electrode line
4. Tubular ceramic
5. Electrode

The below image illustrates the parts present in a metal oxide gas sensor



The purpose of each of these elements is as below:

**Gas sensing layer:** It is the main component in the sensor which can be used to sense the variation in the concentration of the gases and generate the change in electrical resistance. The gas sensing layer is basically a chemiresistor which changes its resistance value based on the

The concentration of particular gas in the environment. Here the sensing element is made up of a Tin Dioxide (SnO2) which is, in general, has excess electrons (donor element). So whenever toxic gases are being detected the resistance of the element changes and the current flown through it varies which represents the change in concentration of the gases.

**Heater coil:** The purpose of the heater coil is to burn-in the sensing element so that the sensitivity and efficiency of the sensing element increases. It is made of Nickel-Chromium which has a high melting point so that it can stay heated up without getting melted.

**Electrode line:** As the sensing element produces a very small current when the gas is detected it is more important to maintain the efficiency of carrying those small currents. So Platinum wires come into play where it helps in moving the electrons efficiently.

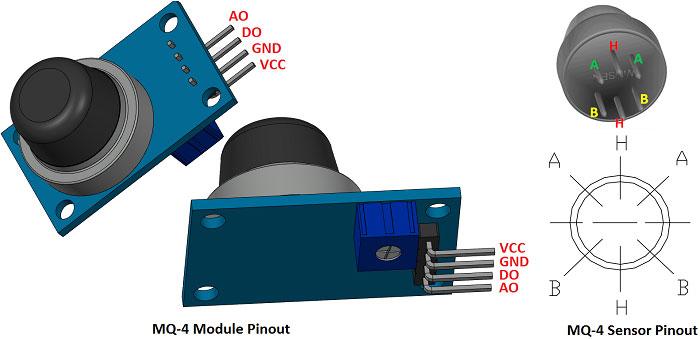
**Electrode:** It is a junction where the output of the sensing layer is connected to the Electrode line. So that the output current can flow to the required terminal. An electrode here is made of Gold (Au –Aurum) which is a very good conductor.

**Tubular ceramic:** In between the Heater coil and Gas sensing layer, the tubular ceramic exists which is made of Aluminum oxide (Al2O3). As it has high melting point, it helps in maintaining the burn-in (preheating) of the sensing layer which gives the high sensitivity for the sensing layer to get efficient output current.

**Mesh over the sensing element:** In order to protect the sensing elements and the setup, a metal mesh is used over it, which is also used to avoid/hold the dust particles entering into the mesh and prevent damaging the gas sensing layer from corrosive particles.

**How** **to use a Gas sensor?**

A basic gas sensor has 6 terminals in which 4 terminals (A, A, B, B) acts input or output and the remaining 2 terminals (H, H) are for heating the coil. Of these 4 terminals, 2 terminals from each side can be used as either input or output (these terminals are reversible as shown in the circuit diagram) and vice versa.



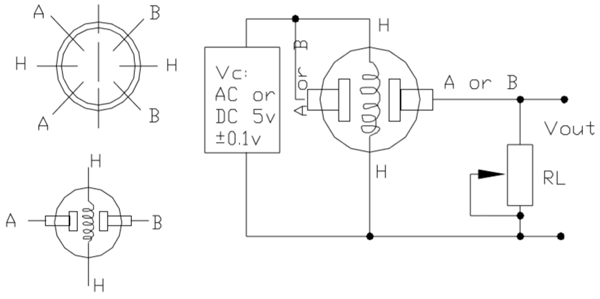
These sensors are normally available as modules (shown right), these modules consist of the gas sensor and a [comparator IC](https://components101.com/tags/voltage-comparator). Now let’s see the pin description of the gas sensor module which we will generally use with an Arduino. The gas sensor module basically consists of 4 terminals

* Vcc – Power supply
* GND – Power supply
* Digital output – This pin gives an output either in logical high or logical low (0 or 1) that means it displays the presence of any toxic or combustible gases near the sensor.
* Analog output – This pin gives an output continuous in voltage which varies based on the concentration of gas that is applied to the gas sensor.

As discussed earlier the output of a gas sensor alone will be very small (in mV) so an external circuit has to be used in order to get a digital high low output from the sensor. For this purpose, a comparator ([LM393](https://components101.com/ics/lm393-low-offset-voltage-dual-comparators)), [adjustable potentiometer](https://components101.com/potentiometer), [some resistors](https://components101.com/resistors) and [capacitors](https://components101.com/capacitors) are used.

The purpose of LM393 is to get the output from the sensor, compare it with a reference voltage and display whether the output is logically high or not. Whereas the purpose of the potentiometer is to set the required threshold value of the gas above which the digital output pin should go high.

The below diagram shows the basic circuit diagram of a gas sensor in a gas sensor module



Here A and B are the input and output terminals (these are reversible - means any of the paired terminals can be used as input or output) and H is the Heater coil terminal. The purpose of the variable resistor is to adjust the output voltage and to maintain high sensitivity.

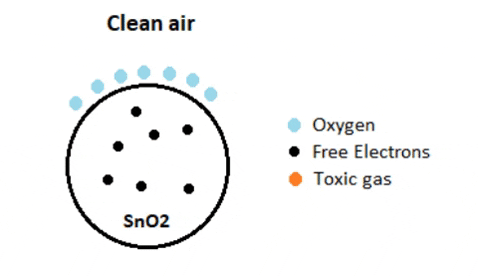
If no input voltage is applied to the heater coil, then the output current will be very less (which is negligible or approximately 0). When sufficient voltage is applied to the input terminal and heater coil, the sensing layer wakes up and is ready to sense any combustible gases nearby it. Initially let’s assume that there is no toxic gas near the sensor, so the resistance of the layer doesn’t change and the output current and voltage are also unchanged and are negligible (approximately 0).

Now let’s assume that there is some toxic gas nearby. As the heater coil is pre-heated it is now easy to detect any combustible gases. When the sensing layer interacts with the gases, the resistance of the material varies and the current flowing through the circuit also varies. This change in variation can be then observed at the load resistance (RL).

The value of load resistance (RL) can be anywhere from 10KΩ to 47KΩ. The exact value of the load resistance can be selected by calibrating with the known concentration of the gas. If low load resistance is selected then the circuit has less sensitivity and if high load resistance is selected then the circuit has high sensitivity.

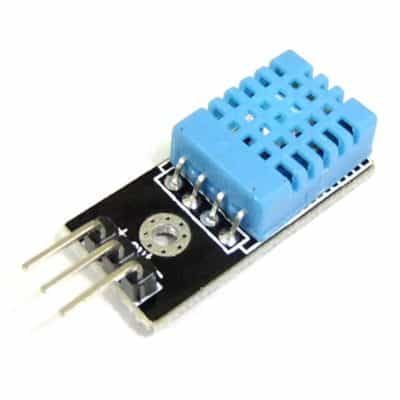
**Working**

The ability of a Gas sensor to detect gases depends on the chemiresister to conduct current. The most commonly used chemiresistor is Tin Dioxide (SnO2) which is an n-type semiconductor that has free electrons (also called as donor). Normally the atmosphere will contain more oxygen than combustible gases. The oxygen particles attract the free electrons present in SnO2 which pushes them to the surface of the SnO2. As there are no free electrons available output current will be zero. The below gif shown the oxygen molecules (blue color) attracting the free electrons (black color) inside the SnO2 and preventing it from having free electrons to conduct current.



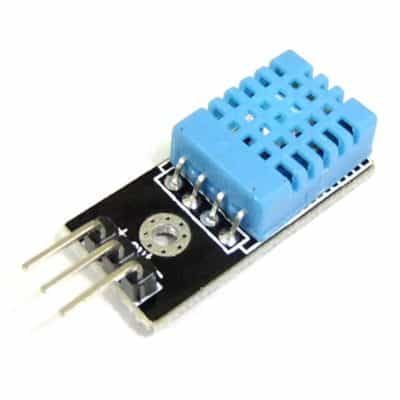
**DHT11 SENSOR:**

This sensor is used here to monitor the humidity variation of the environment where the crops are cultivated. This is a digital sensor and measures the humidity value in percentage format.



**DHT11 Sensor**

DHT11 humidity and temperature sensor is available as a sensor and as a module. The difference between this sensor and module is the pull-up resistor and a power-on LED. DHT11 is a relative humidity sensor.  To measure the surrounding air this sensor uses a [thermistor](https://www.elprocus.com/introduction-to-thermistor-types-with-its-workings-and-applications/) and a capacitive humidity sensor.



DHT11 is a low-cost digital sensor for sensing temperature and humidity.  This sensor can be easily interfaced with any micro-controller such as Arduino, Raspberry Pi etc… to measure humidity and temperature instantaneously.

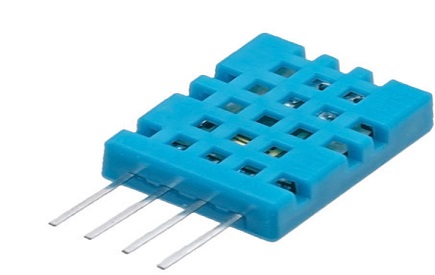
**Working**

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature.  The humidity sensing [capacitor](https://www.elprocus.com/construction-of-capacitor-with-working/) has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

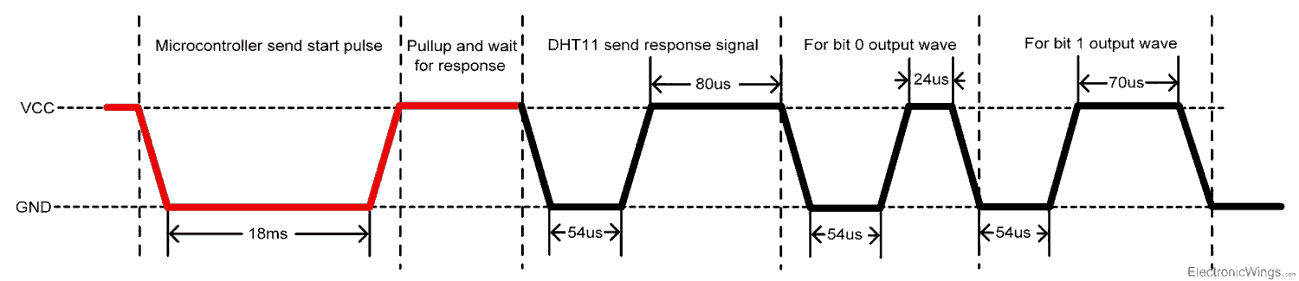
The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. Humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz .i.e. it gives one reading for every second.  DHT11 is small in size with operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA.

DHT11 sensor has four pins- VCC, GND, Data Pin and a not connected pin. A pull-up resistor of 5k to 10k ohms is provided for communication between sensor and micro-controller.



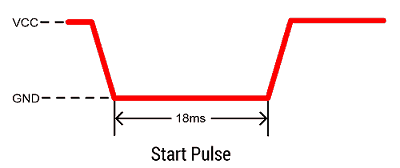
**6.3 HUMIDITY (DHT11):**

* DHT11 uses only one wire for communication. The voltage levels with certain time value defines the logic one or logic zero on this pin.
* The communication process is divided in three steps, first is to send request to DHT11 sensor then sensor will send response pulse and then it starts sending data of total 40 bits to the microcontroller.



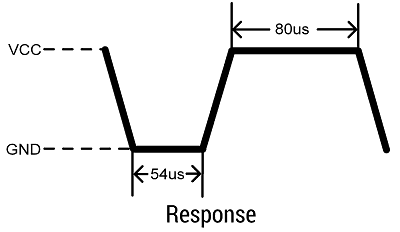
**Communication process**

Start pulse (Request)

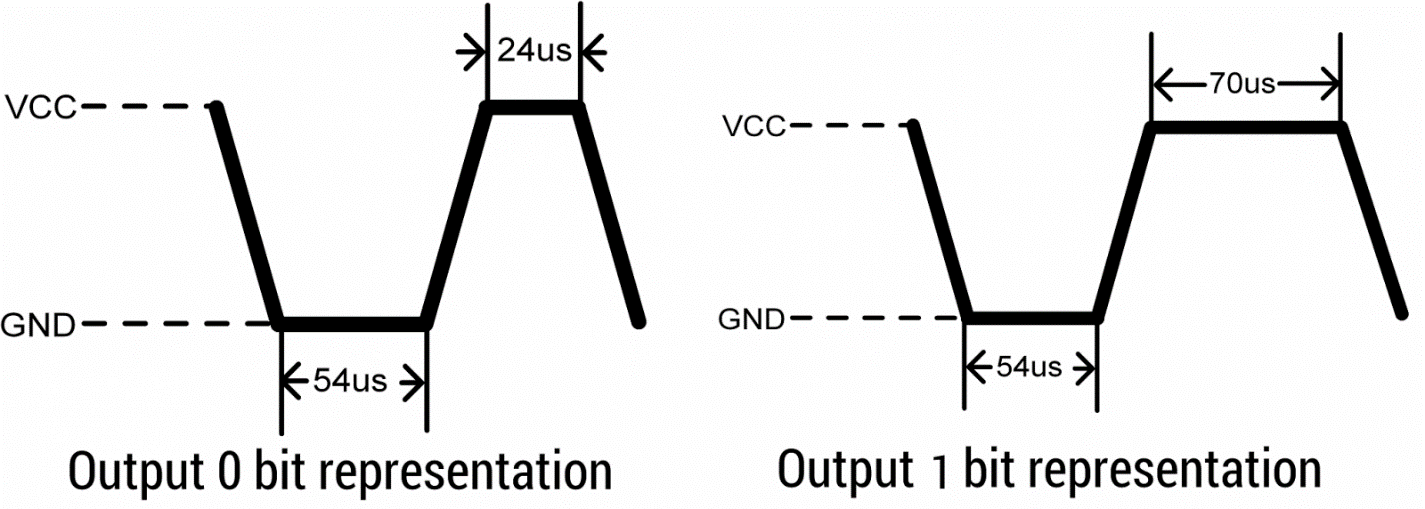


* To start communication with DHT11, first we should send the start pulse to the DHT11 sensor.
* To provide start pulse, pull down (low) the data pin minimum 18ms and then pull up, as shown in diag.

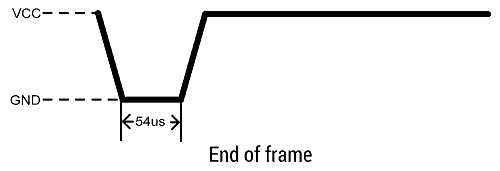
**Response**



* After getting start pulse from, DHT11 sensor sends the response pulse which indicates that DHT11 received start pulse.
* The response pulse is low for 54us and then goes high for 80us.



* After sending the response pulse, DHT11 sensor sends the data, which contains humidity and temperature value along with checksum.
* The data frame is of total 40 bits long, it contains 5 segments (byte) and each segment is 8-bit long.
* In these 5 segments, first two segments contain humidity value in decimal integer form. This value gives us Relative Percentage Humidity. 1st 8-bits are integer part and next 8 bits are fractional part.
* Next two segments contain temperature value in decimal integer form. This value gives us temperature in Celsius form.
* Last segment is the checksum which holds checksum of first four segments.
* **End of frame**



* After sending 40-bit data, DHT11 sensor sends 54us low level and then goes high. After this DHT11 goes in sleep mode.

**PELTIER PANEL:**

A Peltier panel, or thermoelectric cooler, works by using the Peltier effect, where a direct current flowing through junctions of different materials causes one side to heat up and the other to cool down.

Here's a more detailed explanation:

The Peltier Effect:

When a direct current (DC) flows through the junctions of two different materials, one side of the junction absorbs heat (cooling) and the other side releases heat (heating).

Peltier Panel Construction:

A Peltier panel consists of multiple pairs of p-type and n-type semiconductor materials connected in series, with copper bridges acting as electrical connections.

How it Works:

When a DC current flows through the panel, electrons move from the cold side to the hot side.

As electrons move, they absorb heat from the cold side, causing it to cool down.

The electrons then release the absorbed heat on the hot side, causing it to heat up.

Materials:

Bismuth telluride (Bi2Te3) is a common material used in Peltier panels due to its high performance in the temperature range of -100 to +200°C.

Applications:

Peltier panels are used in various applications, including cooling electronic components, small refrigerators, and other applications where compact and efficient cooling is needed.



**FAN:**

Fan sensors, often embedded tachometers or pulse sensors, detect and report the rotational speed of a fan motor using a pulsed signal, enabling closed-loop feedback control and monitoring for optimal cooling performance.

Here's a more detailed explanation:

Tachometer/Pulse Sensors:

These sensors provide information about the fan's speed in RPM (revolutions per minute) through a pulsed signal.

**Function:**

The sensor detects the fan motor's rotational speed and sends this data as a series of pulses, which can be interpreted by a controller to adjust fan speed or monitor its performance.

Types:

Embedded Tachometer: This type is integrated into the fan motor and provides a continuous signal indicating the fan's speed.

Locked Rotor Signal: This signal indicates whether the fan motor is rotating or stopped, helping to detect potential issues.

Applications:

Closed-Loop Control: The sensor's output allows for precise control of fan speed based on temperature or other factors.

Monitoring: The sensor data helps monitor fan performance, detect malfunctions (like a stopped fan), and ensure optimal cooling.

Example:

In a computer, a fan sensor can detect if a fan is spinning too slowly or has stopped, triggering an alert or shutting down the system to prevent overheating.

4-Wire Fans:

These fans have a PWM (Pulse Width Modulation) input for speed control, a tachometer signal for feedback, and power and ground connections.

3-Wire Fans:

These fans have power, ground, and tachometer signals.



**RELAY**

**RELAY**

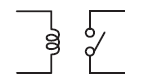
Relays are electrically operated switches that open and close the circuits by receiving electrical signals from outside sources. Some people may associate “relay” with a racing competition where members of the team take turns passing batons to complete the race.  
The “relays” embedded in electrical products work in a similar way; they receive an electrical signal and send the signal to other equipment by turning the switch on and off.

For example, when you push the button on a TV remote to watch TV, it sends an electrical signal to the “relay” inside the TV, turning the main power ON. There are various types of relays used in many applications to control different amounts of currents and number of circuits.

## Electrical Relay Types and Classification

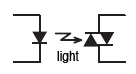
Relay technology can be divided into two main categories: Movable contacts (mechanical relay) and no movable contacts ([MOS FET relay](https://components.omron.com/us-en/products/relays/mosfet-relays), [solid state relay](https://components.omron.com/us-en/products/relays/industrial-relays)).

### Movable contacts ( Mechanical Relay )



This type of relay has contacts that are mechanically actuated to open/close by a magnetic force to switch signals, currents and voltages ON or OFF.

### No movable contacts ( MOS FET relay, Solid State Relay )



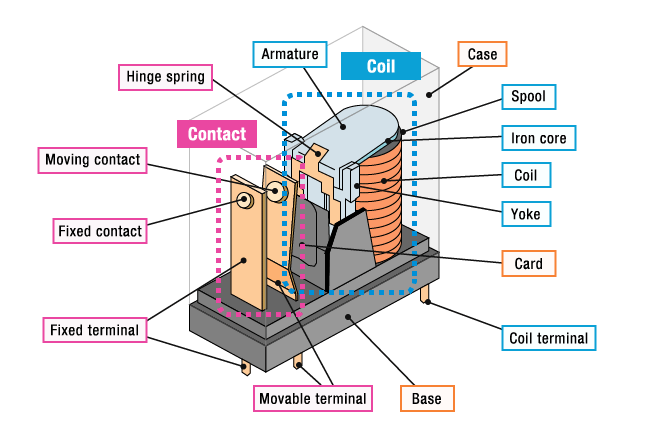
Unlike mechanical relays, this type of relay has no moving contacts but instead employs semiconductor and electrical switching elements such as triac and MOS FET. By the operation of these electronic circuits, signals, currents and voltages are switched ON or OFF electronically.

## Electrical relay Structure and Operating Principles

### 1. Mechanical Relay

#### Basic structure of mechanical relays

Relay consists of a coil, which receives an electric signal and converts it to a mechanical action and contacts that open and close the electric circuit.



****

Relays can be of different types like electromechanical, [solid state](https://www.electronicshub.org/solid-state-relay/). Electromechanical relays are frequently used. Let us see the internal parts of this relay before knowing about it working. Although many different types of relay were present, their working is same.

Every electromechanical relay consists of an consists of an

1. Electromagnet
2. Mechanically movable contact
3. Switching points and
4. Spring

Electromagnet is constructed by wounding a copper coil on a metal core. The two ends of the coil are connected to two pins of the relay as shown. These two are used as DC supply pins.



Generally two more contacts will be present, called as switching points to connect high ampere load. Another contact called common contact is present in order to connect the switching points.

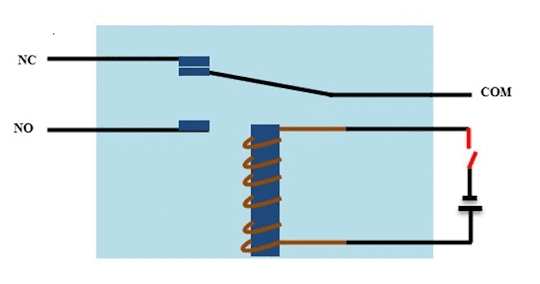
These contacts are named as normally open (NO),normally closed(NC) and common(COM) contacts.

We can use a Relay either in a AC circuit or a DC Circuit. In case of AC relays, for every current zero position, the relay coil gets demagnetized and hence there would be a chance of continues breaking of the circuit.

So, AC relays are constructed with special mechanism such that continuous magnetism is provided in order to avoid above problem. Such mechanisms include electronic circuit arrangement or shaded coil mechanism.

**How a Relay Works?**

The following animation shows a simplified working of a relay.



* Relay works on the principle of electromagnetic induction.
* When the electromagnet is applied with some current, it induces a magnetic field around it.
* Above image shows working of the relay. A switch is used to apply DC current to the load.
* In the relay, Copper coil and the iron core acts as electromagnet.
* When the coil is applied with DC current, it starts attracting the contact as shown. This is called energizing of relay.
* When the supply is removed it retrieves back to the original position. This is called De energizing of relay.

There are also such relays, whose contacts are initially closed and opened when there is supply i.e. exactly to opposite to the above shown relay.

Solid state relays will have sensing element to sense the input voltage and switches the output using opto-coupling.

**Relay Contact Types**

As we have seen that relay is a switch. The terminology “Poles and throws” is also applicable for relay. Depending on the number of contacts and number of circuits it switches relays can be classified.

Before we know about this classification of contacts we have to know the poles and throws of a relay switch.

**Poles and Throws**

Relays can switch one or more circuits. Each switch in relay is referred as pole. Number of circuits a relay connects is indicated by throws.

Depending on the poles and throws, relays are classified into

* Single Pole Single Throw
* Single Pole Double Throw
* Double Pole Single Throw
* Double Pole Double Throw

**Single Pole Single Throw**

A single pole single throw relay can control one circuit and can be connected to one output. It is used for the applications which require only ON or OFF state.

**Single Pole Double Throw**

A single pole double throw relay connects one input circuit to one of the two outputs. This relay is also called as changeover relay.

Though the SPDT has two output positions, it may consist of more than two throws depends on the configuration and requirement of the application.

**Double pole single throw**

A double pole single throw relay has two poles and single throw and it can be used to connect two terminals of a single circuit at a time. For example, this relay is used for connecting both phase and neutral terminals to the load at a time.

**Double pole double throw**

A DPDT (double pole double throw) relay has two poles and two throws for each pole. In motor direction control, these are used for phase or polarity reversal.

The switching action between contacts for all these relays is performed when the coil get energized.

**Types of Relays**

Relays can be classified into different types depending on their functionality, structure, application etc. We listed out some of the common types of relays here.

* Electromagnetic
* Latching
* Electronic
* Non-Latching
* Reed
* High-Voltage
* Small Signal
* Time Delay
* Multi-Dimensional
* Thermal
* Differential
* Distance
* Automotive
* Frequency
* Polarized
* Rotary
* Sequence
* Moving Coil
* Buchholz
* Safety
* Supervision
* Ground Fault

**Relay Applications:**

Relays are used to protect the electrical system and to minimize the damage to the equipment connected in the system due to over currents/voltages. The relay is used for the purpose of protection of the equipment connected with it.

These are used to control the high voltage circuit with low voltage signal in applications audio amplifiers and some types of modems.

These are used to control a high current circuit by a low current signal in the applications like starter solenoid in automobile. These can detect and isolate the faults that occurred in power transmission and distribution system. Typical application areas of the relays include

* Lighting control systems
* Telecommunication
* Industrial process controllers
* Traffic control
* Motor drives control
* Protection systems of electrical power system
* Computer interfaces
* Automotive
* Home appliances

**How to Test a Relay?**

Since they are electromechanical devices, relays can wear out eventually and stop working over time. But there are few   techniques to test if a relay is working or not. These techniques include:

* Testing a Relay with a Multi meter
* Build a simple circuit to test the Relay

Use a DC Power Supply to see whether a relay is functioning properly.

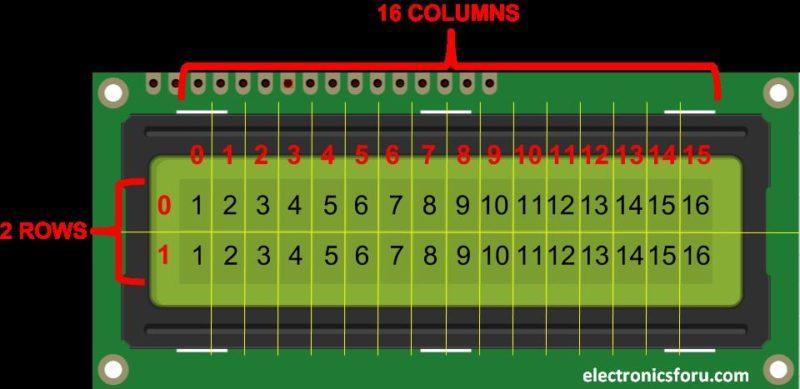
**LCD**

**I2C LCD:**

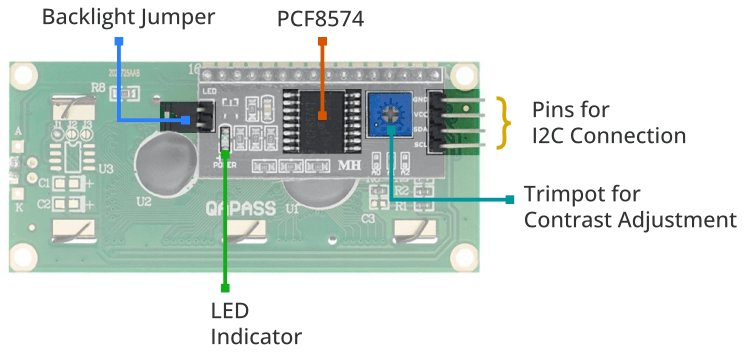
LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A 16×2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over seven segments and other multi segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on.

A 16×2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5×7 pixel matrix.

The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD. Click to learn more about internal structure of an LCD.

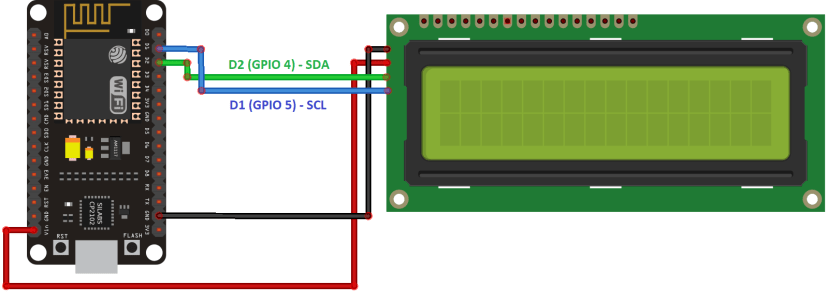


At the center of this adapter, there is an 8-bit I/O expander chip – PCF8574. It takes the I2C data from the MCU (Arduino) and converts it into serial data required for an LCD display. At one side the I2C LCD adapter has four pins that can be connected to Arduino or any microcontroller that supports the I2C communication protocol. On another side, it has 16 pins that are connected to the LCD display. There are two header pins to control the backlight of the LCD display. One pin supplies a 5v power and another pin is for the backlight LED. These two pins are connected together by default. So the backlight will be always on. You can remove the jumper to turn off the backlight LED or you can use a potentiometer in between these two pins to control the intensity of the backlight LED.



**Working:**

The I2C LCD operates by connecting to a microcontroller through two wires: SDA (Serial Data) and SCL (Serial Clock). It communicates using the I2C protocol, receiving commands and data from the microcontroller. An embedded controller within the LCD module interprets and processes this data to control the display, including showing characters, numbers, and symbols while efficiently using minimal hardware pins. This streamlined communication method simplifies wiring and hardware requirements, making it popular in various embedded systems.



**I2C LCD pinout:**

|  |  |
| --- | --- |
| **Pin Name** | **Description** |
| VCC | Power supply (usually +5V) |
| GND | Ground |
| SDA | I2C Serial Data Line |
| SCA | I2C Serial Clock Line |

**Features of 16×2 LCD module**

Operating Voltage is 4.7V to 5.3V

Current consumption is 1mA without backlight

Alphanumeric LCD display module, meaning can display alphabets and numbers

Consists of two rows and each row can print 16 characters.

Each character is build by a 5×8 pixel box

Can work on both 8-bit and 4-bit mode

It can also display any custom generated characters

Available in Green and Blue Backlight

**Advantages**

LCD’s consumes less amount of power compared to CRT and LED

LCD’s are consist of some microwatts for display in comparison to some mill watts for LED’s

LCDs are of low cost

Provides excellent contrast

LCD’s are thinner and lighter when compared to cathode ray tube and LED

**Disadvantages**

Require additional light sources

Range of temperature is limited for operation

Low reliability

Speed is very low

LCD’s need an AC drive

**Applications**

Liquid crystal technology has major applications in the field of science and engineering as well on electronic devices.

Liquid crystal thermometer

Optical imaging

The liquid crystal display technique is also applicable in visualization of the radio frequency waves in the waveguide

Used in the medical applications.

**SOFTWARE REQUIREMENTS:**

**SOFTWARE DESCRIPTION**

**Arduino Software (IDE)**

[Writing Sketches](https://www.arduino.cc/en/Guide/Environment#toc1)

[File](https://www.arduino.cc/en/Guide/Environment#toc2)

[Edit](https://www.arduino.cc/en/Guide/Environment#toc3)

[Sketch](https://www.arduino.cc/en/Guide/Environment#toc4)

[Tools](https://www.arduino.cc/en/Guide/Environment#toc5)

[Help](https://www.arduino.cc/en/Guide/Environment#toc6)

[Sketchbook](https://www.arduino.cc/en/Guide/Environment#toc7)

[Tabs, Multiple Files, and Compilation](https://www.arduino.cc/en/Guide/Environment#toc8)

[Uploading](https://www.arduino.cc/en/Guide/Environment#toc9)

[Libraries](https://www.arduino.cc/en/Guide/Environment#toc10)

[Third-Party Hardware](https://www.arduino.cc/en/Guide/Environment#toc11)

[Serial Monitor](https://www.arduino.cc/en/Guide/Environment#toc12)

[Preferences](https://www.arduino.cc/en/Guide/Environment#toc13)

[Language Support](https://www.arduino.cc/en/Guide/Environment#toc14)

[Boards](https://www.arduino.cc/en/Guide/Environment#toc15)

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

**Writing Sketches**

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension. ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.

|  |  |
| --- | --- |
| https://www.arduino.cc/en/uploads/Guide/play.png | Verify  Checks your code for errors compiling it. |
| https://www.arduino.cc/en/uploads/Guide/export.png | Upload  Compiles your code and uploads it to the configured board. See [uploading](https://www.arduino.cc/en/Guide/Environment" \l "uploading)below for details.  Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer" |
| https://www.arduino.cc/en/uploads/Guide/new.png | New  Creates a new sketch. |
|  | https://www.arduino.cc/en/uploads/Guide/open.png |

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

**File**

* New   
  Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
* Open   
  Allows to load a sketch file browsing through the computer drives and folders.
* Open Recent   
  Provides a short list of the most recent sketches, ready to be opened.
* Sketchbook   
  Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
* Examples   
  Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
* Close   
  Closes the instance of the Arduino Software from which it is clicked.
* Save   
  Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
* Save as...   
  Allows to save the current sketch with a different name.
* Page Setup   
  It shows the Page Setup window for printing.
* Print   
  Sends the current sketch to the printer according to the settings defined in Page Setup.
* Preferences   
  Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
* Quit   
  Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

**Edit**

* Undo/Redo   
  Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
* Cut   
  Removes the selected text from the editor and places it into the clipboard.
* Copy   
  Duplicates the selected text in the editor and places it into the clipboard.
* Copy for Forum   
  Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
* Copy as HTML   
  Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
* Paste   
  Puts the contents of the clipboard at the cursor position, in the editor.
* Select All   
  Selects and highlights the whole content of the editor.
* Comment/Uncomment   
  Puts or removes the // comment marker at the beginning of each selected line.
* Increase/Decrease Indent   
  Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
* Find   
  Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
* Find Next   
  Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
* Find Previous   
  Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

**Sketch**

* Verify/Compile   
  Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
* Upload   
  Compiles and loads the binary file onto the configured board through the configured Port.
* Upload Using Programmer   
  This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a Tools -> Burn Bootloader command must be executed.
* Export Compiled Binary   
  Saves a .hex file that may be kept as archive or sent to the board using other tools.
* Show Sketch Folder   
  Opens the current sketch folder.
* Include Library   
  Adds a library to your sketch by inserting #include statements at the start of your code. For more details, see [libraries](https://www.arduino.cc/en/Guide/Environment#libraries) below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
* Add File...   
  Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right side o the toolbar.

**Tools**

* Auto Format   
  This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
* Archive Sketch   
  Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
* Fix Encoding & Reload   
  Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
* Serial Monitor   
  Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.
* Board   
  Select the board that you're using. See below for [descriptions of the various boards](https://www.arduino.cc/en/Guide/Environment#boards).
* Port   
  This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
* Programmer   
  For selecting a harware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're [burning a bootloader](https://www.arduino.cc/en/Tutorial/Bootloader) to a new microcontroller, you will use this.
* Burn Bootloader   
  The items in this menu allow you to burn a [bootloader](https://www.arduino.cc/en/Hacking/Bootloader) onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino or Genuino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the Boards menu before burning the bootloader on the target board. This command also set the right fuses.

**Help**

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

* Find in Reference   
  This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

**Sketchbook**

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the File > Sketchbook menu or from the Open button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the Preferences dialog.

Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the pde extension. You may still open.pde named files in version 1.0 and later, the software will automatically rename the extension to ino.

**Tabs, Multiple Files, and Compilation**

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

**Uploading**

Before uploading your sketch, you need to select the correct items from the Tools > Board and Tools > Port menus. The [boards](https://www.arduino.cc/en/Guide/Environment#boards) are described below. On the Mac, the serial port is probably something like /dev/tty.usbmodem241 (for an Uno or Mega2560 or Leonardo) or /dev/tty.usbserial-1B1 (for a Duemilanove or earlier USB board), or /dev/tty.USA19QW1b1P1.1 (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be /dev/ttyACMx , /dev/ttyUSBx or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the Upload item from the Sketch menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

**Libraries**

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the Sketch > Import Library menu. This will insert one or more #include statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its #includestatements from the top of your code.

There is a [list of libraries](https://www.arduino.cc/en/Reference/Libraries) in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these [instructions for installing a third-party library](https://www.arduino.cc/en/Guide/Libraries).

To write your own library, see [this tutorial](https://www.arduino.cc/en/Hacking/LibraryTutorial).

**Third-Party Hardware**

Support for third-party hardware can be added to the hardware directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the hardware directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.)

**Serial Monitor**

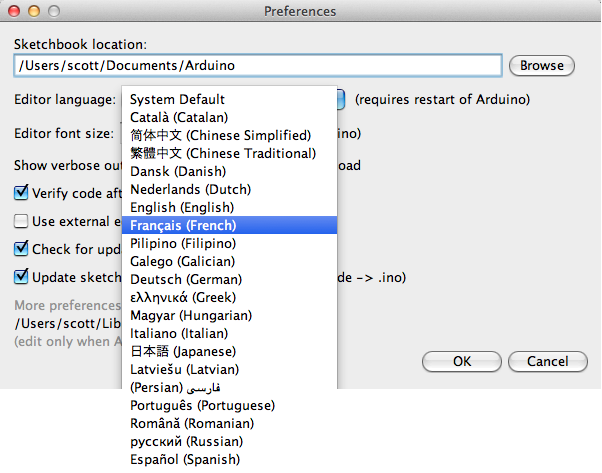
Displays serial data being sent from the Arduino or Genuino board (USB or serial board). To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down that matches the rate passed to Serial.begin in your sketch. Note that on Windows, Mac or Linux, the Arduino or Genuino board will reset (rerun your sketch execution to the beginning) when you connect with the serial monitor.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the [interfacing page](http://www.arduino.cc/playground/Main/Interfacing) for details).

**Preferences**

Some preferences can be set in the preferences dialog (found under the Arduino menu on the Mac, or File on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

**Language Support**



Since version 1.0.1 , the Arduino Software (IDE) has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.)

If you would like to change the language manually, start the Arduino Software (IDE) and open the Preferences window. Next to the Editor Language there is a dropdown menu of currently supported languages. Select your preferred language from the menu, and restart the software to use the selected language. If your operating system language is not supported, the Arduino Software (IDE) will default to English.

**Boards**

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader. You can find a comparison table between the various boards [here](https://www.arduino.cc/en/Products/Compare).

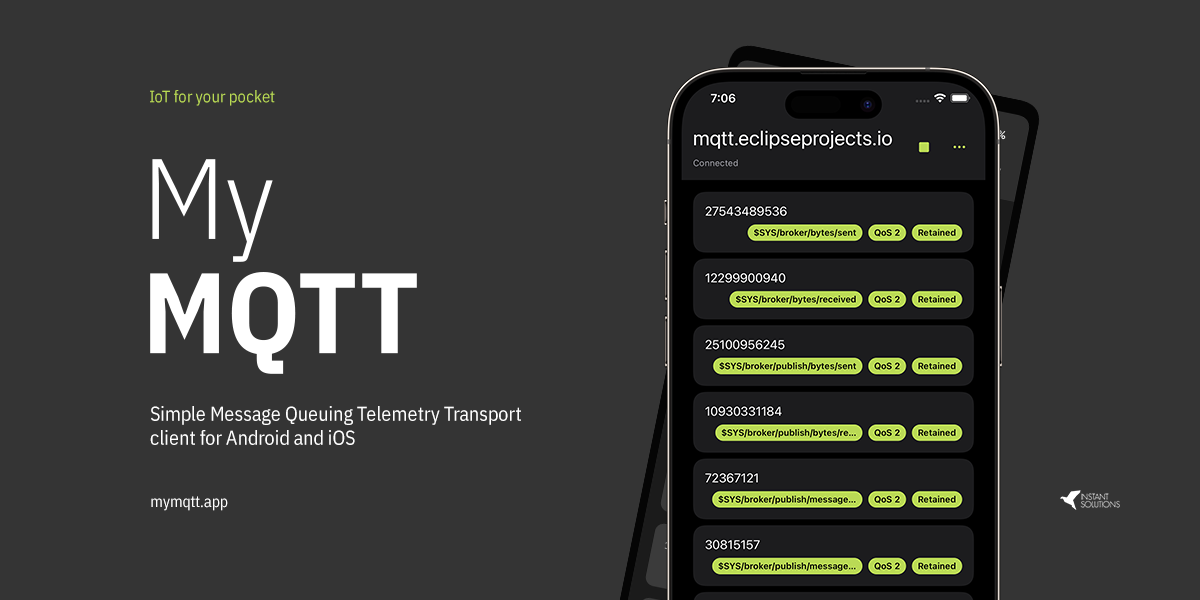
Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The [Boards Manager](https://www.arduino.cc/en/Guide/Cores) included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

* Arduino Yùn   
  An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
* Arduino/Genuino Uno   
  An ATmega328 running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
* Arduino Diecimila or Duemilanove w/ ATmega168   
  An ATmega168 running at 16 MHz with auto-reset.
* Arduino Nano w/ ATmega328   
  An ATmega328 running at 16 MHz with auto-reset. Has eight analog inputs.
* Arduino/Genuino Mega 2560   
  An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
* Arduino Mega   
  An ATmega1280 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
* Arduino Mega ADK   
  An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
* Arduino Leonardo   
  An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
* Arduino/Genuino Micro   
  An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
* Arduino Esplora   
  An ATmega32u4 running at 16 MHz with auto-reset.
* Arduino Mini w/ ATmega328  
  An ATmega328 running at 16 MHz with auto-reset, 8 Analog In, 14 Digital I/O and 6 PWM.
* Arduino Ethernet   
  Equivalent to Arduino UNO with an Ethernet shield: An ATmega328 running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
* Arduino Fio   
  An ATmega328 running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328, 6 Analog In, 14 Digital I/O and 6 PWM.
* Arduino BT w/ ATmega328   
  A Tmega328 running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board bluetooth module, 6 Analog In, 14 Digital I/O and 6 PWM..
* Lily Pad Arduino USB   
  An ATmega32u4 running at 8 MHz with auto-reset, 4 Analog In, 9 Digital I/O and 4 PWM.
* LilyPad Arduino   
  An ATmega168 or ATmega132 running at 8 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
* Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328   
  An ATmega328 running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328; 6 Analog In, 14 Digital I/O and 6 PWM.
* Arduino NG or older w/ ATmega168 .

**IOT** **MQTT (**M**essage** Q**ueuing** T**elemetry**T**ransport)**

**Introduction**

MQTT is a lightweight publish/subscribe messaging protocol. It is useful for use with low power sensors, but is applicable to many scenarios



**Publish/Subscribe**

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, common interface for everything to connect to.

**Topics/Subscriptions**

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem. For example, multiple computers may all publish their hard drive temperature information on the following topic, with their own computer and hard drive name being replaced as appropriate:

**Client**

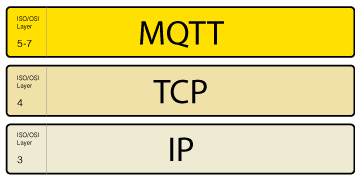
When we talk about a client, we almost always mean an [**MQTT client**](https://www.hivemq.com/blog/seven-best-mqtt-client-tools/). Both publishers and subscribers are MQTT clients. The publisher and subscriber labels refer to whether the client is currently publishing messages or subscribing to messages (publish and subscribe functionality can also be implemented in the same MQTT client). **An MQTT client is any device (from a micro controller up to a full-fledged server) that runs an MQTT library and connects to an**[**MQTT broker**](https://www.hivemq.com/hivemq/)**over a network.** For example, the MQTT client can be a very small, resource-constrained device that connects over a wireless network and has a bare-minimum library. The MQTT client can also be a typical computer running a graphical MQTT client for testing purposes. Basically, any device that speaks MQTT over a TCP/IP stack can be called an MQTT client. The client implementation of the MQTT protocol is very straight forward and streamlined. The ease of implementation is one of the reasons why MQTT is ideally suited for small devices. **MQTT client libraries are available for a huge variety of programming languages. For example, Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, and .NET.** You can see a complete list on the [**MQTT wiki**](https://github.com/mqtt/mqtt.github.io/wiki/libraries).

**Broker**

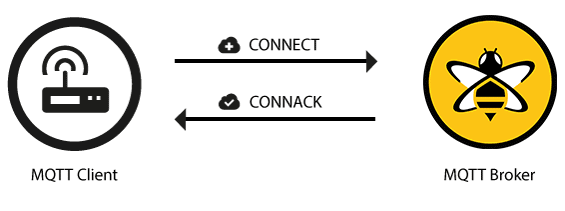
The counterpart of the MQTT client is the MQTT broker. The broker is at the heart of any publish/subscribe protocol. Depending on the implementation, a broker can handle up to thousands of concurrently connected MQTT clients. **The broker is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients.** The broker also holds the sessions of all persisted clients, including subscriptions and missed messages (more [**details**](https://www.hivemq.com/blog/mqtt-essentials-part-7-persistent-session-queuing-messages/)). Another responsibility of the broker is the authentication and authorization of clients. Usually, the broker is extensible, which facilitates custom authentication, authorization, and integration into backend systems. Integration is particularly important because the broker is frequently the component that is directly exposed on the internet, handles a lot of clients, and needs to pass messages to downstream analyzing and processing systems. As discussed in [**a previous post**](https://www.hivemq.com/mqtt-sql-database/), subscribing to all message is not really an option. In brief, the broker is the central hub through which every message must pass. Therefore, **it is important that your broker is highly scalable, integratable into backend systems, easy to monitor, and (of course) failure-resistant.** HiveMQ meets these requirements by using state-of-the-art event-driven network processing, an open plugin system, and standard monitoring providers.

MQTT Connection

The MQTT protocol is based on TCP/IP. Both the client and the broker need to have a TCP/IP stack.



The MQTT connection is always between one client and the broker. Clients never connect to each other directly. To initiate a connection, **the client sends a CONNECT message to the broker. The broker responds with a CONNACK message** and a status code. Once the connection is established, the broker keeps it open until the client sends a disconnect command or the connection breaks.



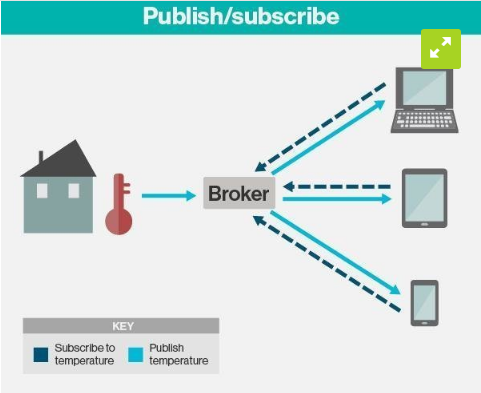
**MQTT connection through a NAT**

In many common use cases, the MQTT client is located behind a router that uses network address translation (NAT) to translate from a private network address (like 192.168.x.x, 10.0.x.x) to a public facing address. As we already mentioned, the MQTT client initiates the connection by sending a CONNECT message to the broker. Because the broker has a public address and keeps the connection open to allow bidirectional sending and receiving of messages (after the initial CONNECT), there is no problem at all with clients that are located behind a NAT.

**Client initiates connection with the CONNECT message**

Now let’s look at the [**MQTT CONNECT**](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028) command message. To initiate a connection, the client send s a command message to the broker. If this CONNECT message is malformed (according to the MQTT specification) or too much time passes between opening a network socket and sending the connect message, the broker closes the connection. This behavior deters malicious clients that can slow the broker down. **A good-natured client sends a connect message with the following content** (among other things):

Some information included in a CONNECT message is probably more interesting to implementers of an MQTT library rather than to users of that library. For all the details, have a look at the [**MQTT 3.1.1 specification**](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html).



**ClientId**

The client identifier (ClientId) **identifies each MQTT client** that connects to an MQTT broker. The broker uses the ClientID to identify the client and the current state of the client.Therefore, this ID should be unique per client and broker. In MQTT 3.1.1 (the current standard), you can send an empty ClientId, if you don’t need a state to be held by the broker. The empty ClientID results in a connection without any state. In this case, the clean session flag must be set to true or the broker will reject the connection.

**Clean Session**

The clean session flag tells the broker whether the client wants to establish a persistent session or not. In a persistent session (CleanSession = false), the broker stores all subscriptions for the client and all missed messages for the client that subscribed with a [**Quality of Service (QoS)**](https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/) level 1 or 2. If the session is not persistent (CleanSession = true), the broker does not store anything for the client and purges all information from any previous persistent session.

**Username/Password**

MQTT can send a **user name and password for client authentication and authorization**. However, if this information isn’t encrypted or hashed (either by implementation or TLS), the password is sent in plain text. We highly recommend the use of user names and passwords together with a secure transport. Brokers like HiveMQ can authenticate clients with an SSL certificate, so no username and password is needed.

**Will Message**

The last will message is part of the Last Will and Testament (LWT) feature of MQTT. **This message notifies other clients when a client disconnects ungracefully.** When a client connects, it can provide the broker with a last will in the form of an MQTT message and topic within the CONNECT message. If the client disconnects ungracefully, the broker sends the LWT message on behalf of the client. You can learn more about LWT in [**part 9**](https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament) of this

**History of the MQTT protocol**

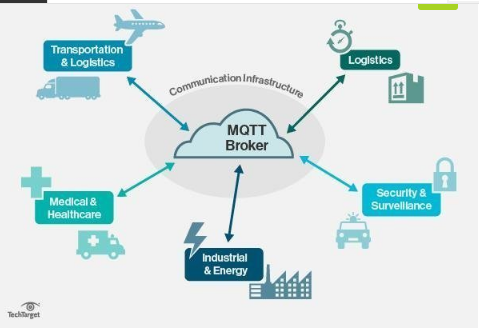
MQTT was created by Dr. Andy Stanford-Clark of IBM and Arlen Nipper of Arcom -- now Eurotech -- in 1999. MQTT was created as a cost-effective and reliable way to connect monitoring devices used in the oil and gas industries to remote enterprise servers. When challenged with finding a way to push data from pipeline sensors in the desert to off-site supervisory control and data acquisition ([SCADA](https://whatis.techtarget.com/definition/SCADA-supervisory-control-and-data-acquisition)) systems, they decided upon a TCP/IP-based publish/subscribe topology that would be [event-driven](https://searchmicroservices.techtarget.com/definition/event-driven-architecture-EDA) to keep satellite link transmission costs down.

Although MQTT is still closely associated with IBM, it is now an open protocol that is overseen by the Organization for the Advancement of Structured Information Standards ([OASIS](https://searchmicroservices.techtarget.com/definition/OASIS-Organization-for-the-Advancement-of-Structured-Information-Standards)).

Though the name suggests it, MQTT is not part of the original IBM [MQSeries](https://whatis.techtarget.com/definition/MQSeries); however, as of version 7.1, it is available in WebSphere MQ. MQTT was previously known as the SCADA protocol, MQ Integrator SCADA Device Protocol (MQIsdp) and WebSphere MQTT (WMQTT), although all these variations have fallen out of use.

**MQTT protocol applications and use cases**

Facebook currently uses MQTT for its Messenger app, not only because the protocol conserves battery power during mobile phone-to-phone messaging, but also because the protocol enables messages to be delivered efficiently in milliseconds (ms), despite inconsistent internet connections across the globe.



Most major cloud services providers, including Amazon Web Services (AWS), Google Cloud, IBM Cloud and Microsoft Azure, support MQTT.

MQTT is well suited to applications using M2M and IoT devices for purposes such as real-time analytics, preventative maintenance and monitoring in environments, including smart homes, healthcare, logistics, industry and manufacturing.

**M****QTT in IoT**

MQTT is one of the most commonly used protocols concerning IoT. MQTT enables resource-constrained IoT devices to send, or publish, information about a given topic to a server that functions as an MQTT message broker. The broker then pushes the information out to those clients that have previously subscribed to the topic. To a human, a topic looks like a hierarchical file path. Clients can subscribe to a specific level of a topic's hierarchy or use a wild-card character to subscribe to multiple levels.

**Competing protocols**

Other transfer protocols that compete with MQTT include the following:

* **Constrained Application Protocol (CoAP)** is another protocol well suited for IoT. CoAP also uses a request/response communication pattern.
* **Advanced Message Queuing Protocol (AMQP)**, like MQTT, uses a publish/subscribe communication pattern.
* **Simple/Streaming Text Oriented Messaging Protocol (STOMP)** is a text-based protocol. However, STOMP does not deal with queues and topics; it uses a send semantic with a destination string.
* **Mosquitto** is an open source MQTT broker.
* **Simple Media Control Protocol (SMCP)** is a CoAP stack that's used in embedded environments. SMCP is also C-based.
* **SSI (Simple Sensor Interface)** is a communications protocol for data transfer between a combination of computers and sensors.
* **Data Distribution Service (DDS)** for real-time systems is a middleware standard that can directly publish or subscribe communications in real time in embedded systems.

**Pros and cons of MQTT**

MQTT has a few distinct advantages and disadvantages when compared to competing protocols. Advantages include the following:

* efficient data transmission and quick to implement due to its being a lightweight protocol;
* low network usage, due to minimized data packets;
* efficient distribution of data;
* successful implementation of remote sensing and control;
* fast and efficient message delivery;
* usage of small amounts of power, which is good for the connected devices; and
* reduction of network bandwidth

**Potential downsides to MQTT include the following:**

* MQTT has slower transmit cycles compared to CoAP.
* MQTT's resource discovery works on flexible topic subscription, whereas CoAP uses a stable resource discovery system.
* MQTT is unencrypted. Instead, it uses TLS/SSL for security encryption.
* It is difficult to create a globally scalable MQTT network.

**MQTT challenges: Security, interoperability and authentication**

Because the MQTT protocol was not designed with security in mind, the protocol has traditionally been used in secure back-end networks for application-specific purposes. MQTT's topic structure can easily form a huge tree, and there's no clear way to divide a tree into smaller logical domains that can be federated. This makes it difficult to create a globally scalable MQTT network because, as the size of the topic tree grows, the complexity increases.

Another negative aspect of MQTT is its lack of interoperability. Because message payloads are binary, with no information as to how they are encoded, problems can arise -- especially in open architectures where different applications from different manufacturers are supposed to work seamlessly with each other.

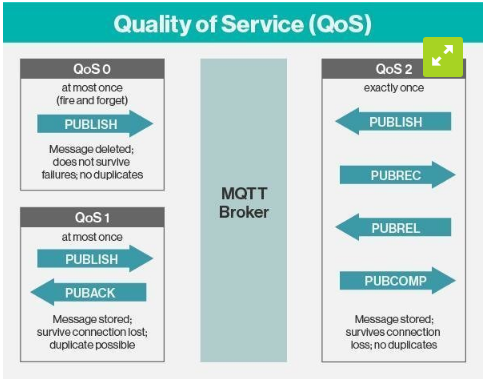
As touched upon previously, MQTT has minimal authentication features built into the protocol. Usernames and passwords are sent in cleartext, and any form of secure use of MQTT must employ SSL/TLS, which, unfortunately, is not a lightweight protocol.

Authenticating clients with client-side certificates is not a simple process, and there's no way in MQTT to control who owns a topic and who can publish information on it, except using proprietary, out-of-band means. This makes it easy to inject harmful messages into the network, either willfully or by mistake.

Furthermore, there's no way for the message receiver to know who sent the original message unless that information is contained in the actual message. Security features that have to be implemented on top of MQTT in a proprietary fashion increase the code footprint and make implementations more difficult.

**Quality of service levels**

QoS refers to an agreement between the sender of a message and the message's recipient. QoS will define the guarantee of delivery in referring to a specific message. QoS acts as a key feature in MQTT, giving the client the ability to choose between three levels of service.



The three different QoS levels determine how the content is managed by the MQTT protocol. Although higher levels of QoS are more reliable, they have more latency and bandwidth requirements, so subscribing clients can specify the highest QoS level they would like to receive.

**Specifications**

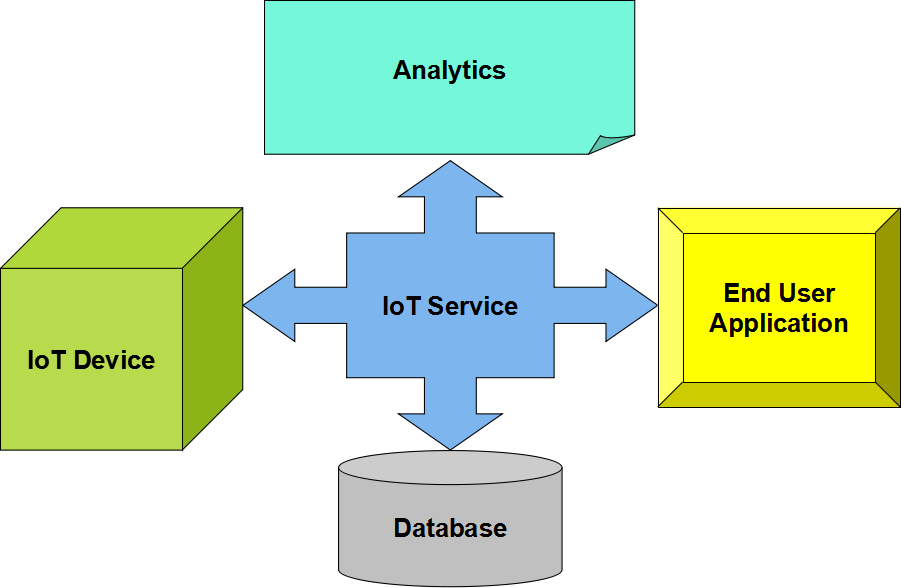
MQTT has different specifications depending on the specific version. Version 5.0 superseded the last version of MQTT, version 3.1.1. Some newer specifications, as defined by OASIS, include the following:

* the use of publish/subscribe message patterns;
* a mechanism that can notify users when abnormal disconnections occur;
* the three levels of message delivery: at most once, at least once and exactly once;
* the minimization of transport overhead and protocol exchanges to reduce network traffic; and
* an agnostic messaging transport referring to the content of the payload.

THING SPEAK:

IOT:

The Internet of Things(IoT) is a system of ‘connected things’. The things generally comprise of an embedded operating system and an ability to communicate with the internet or with the neighbouring things. One of the key elements of a generic IoT system that bridges the various ‘things’ is an IoT service. An interesting implication from the ‘things’ comprising the IoT systems is that the things by themselves cannot do anything. At a bare minimum, they should have an ability to connect to other ‘things’. But the real power of IoT is harnessed when the things connect to a ‘service’ either directly or via other ‘things’. In such systems, the service plays the role of an invisible manager by providing capabilities ranging from simple data collection and monitoring to complex data analytics. The below diagram illustrates where an IoT service fits in an IoT ecosystem:



One such IoT application platform that offers a wide variety of analysis, monitoring and counter- action capabilities is ‘ThingSpeak’. Let us consider ThingSpeak in detail.

**What is ThingSpeak**

ThingSpeak is a platform providing various services exclusively targeted for building IoT applications. It offers the capabilities of real-time data collection, visualizing the collected data in the form of charts, ability to create plugins and apps for collaborating with web services, social network and other APIs. We will consider each of these features in detail below.

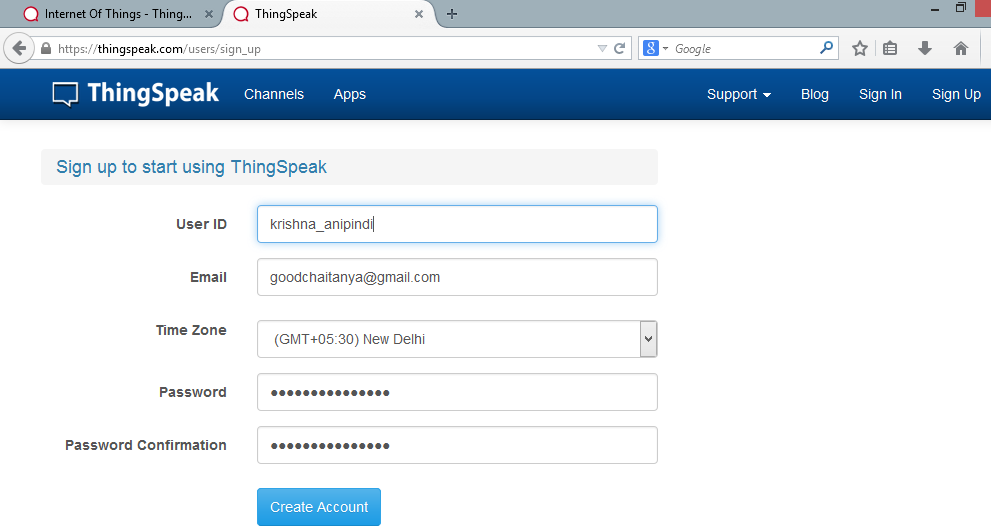
The core element of ThingSpeak is a ‘ThingSpeak Channel’. A channel stores the data that we send to ThingSpeak and comprises of the below elements:

* 8 fields for storing data of any type - These can be used to store the data from a sensor or from an embedded device.
* 3 location fields - Can be used to store the latitude, longitude and the elevation. These are very useful for tracking a moving device.
* 1 status field - A short message to describe the data stored in the channel.

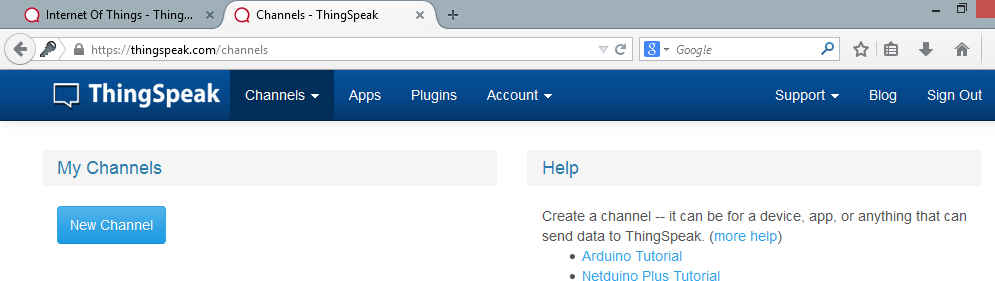
To use ThingSpeak, we need to signup and create a channel. Once we have a channel, we can send the data, allow ThingSpeak to process it and also retrieve the same. Let us start exploring ThingSpeak by signing up and setting up a channel.

**Getting Started**

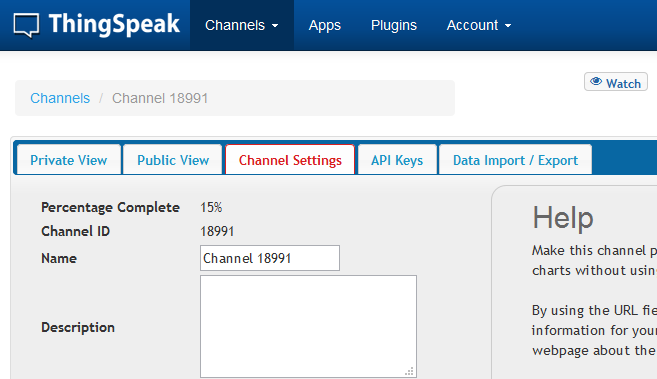
Open <https://thingspeak.com/>and click on the ‘Get Started Now’ button on the center of the page and you will be redirected to the sign-up page(you will reach the same page when you click the ‘Sign Up’ button on the extreme right). Fill out the required details and click on the ‘Create Account’ button.



Now you should see a page with a confirmation that the account was successfully created. The confirmation message disappears after a few seconds and the final page should look as in the below screen:



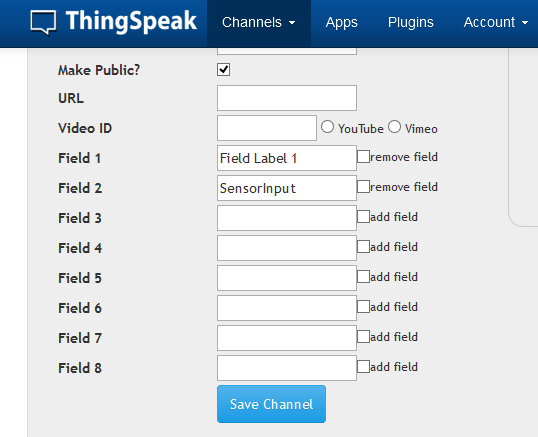
Go ahead and click on ‘New Channel’. You should see a page like the below:



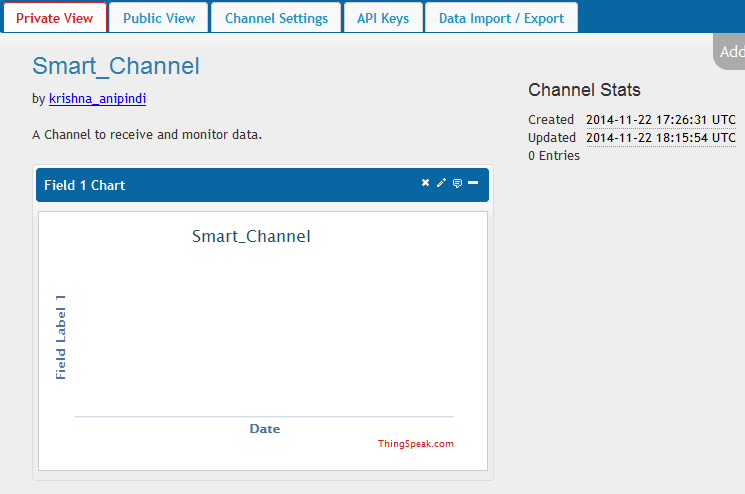
You can change the name to fit your need and you can add a description corresponding to the channel. You can add any other useful description into the metadata field. In the same page, you should see the fields for Latitude, Longitude and Elevation. Also, when you scroll down you should see a check box that says ‘Make Public?’. Let us consider the significance of the various fields and the tabs:

* Latitude, longitude and elevation - These fields correspond to the location of a ‘thing’ and are especially significant for moving things.
* Make Public? - If the channel is made public, anyone can view the channel's data feed and the corresponding charts. If this check box is not checked, the channel is private, which means for every read or write operation, the user has to pass a corresponding API key.
* URL - This can be the URL of your blog or website and if specified, will appear on the public view of the channel.
* Video ID - This is the ID corresponding to your YouTube or Vimeo ID. If specified, the video appears on the public view of the channel.
* Fields 1 to 8 - These are the fields which correspond to the data sent by a sensor or a ‘thing’. A field has to be added before it can be used to store data. By default, Field 1 is added. In case you try posting to fields that you have not added, your request will still be successful, but you will not be able to see the field in the charts and the corresponding data. You can click on the small box before the ‘add field’ text corresponding to each field to add it. Once you click the ‘add field’ box,

a default label name appears in the text box corresponding to each field and the ‘add field’ text changes to ‘remove field’. You can edit the field text that appears by default when a field is added to make more sense. For example, in the below screen, I have modified the text for Field 2 to ‘SensorInput’. To remove a field which is added, just check on the ‘remove field’ box. Once you click this, the text ‘remove field’ changes back to ‘add field’ and the corresponding field text is cleared.



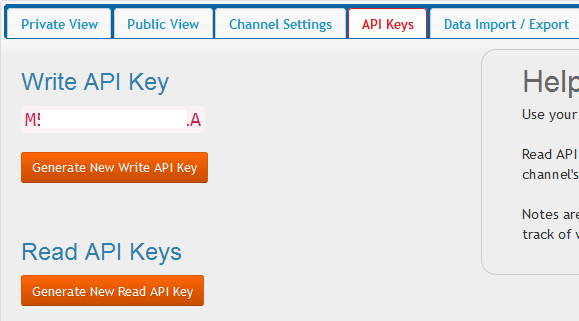
Once you have edited the fields, click on ‘Save Channel’ button. You should now see a page like the below in which the ‘Private View’ tab is defaulted:



The Private View shows a chart corresponding to each of the fields that we have added. Now click on the ‘Public View’ tab. This should look exactly similar to the what we see in the ‘Private View’ tab since our channel is public. In case your channel is not public('make public' check box not checked in the ‘channel settings’ tab), the public view tab shows a message that ‘This channel is not public’.

Now click on the ‘API Keys’ tab. You should see a screen similar to the below. The write API key is used for sending data to the channel and the read API key(s) is used to read the channel data.

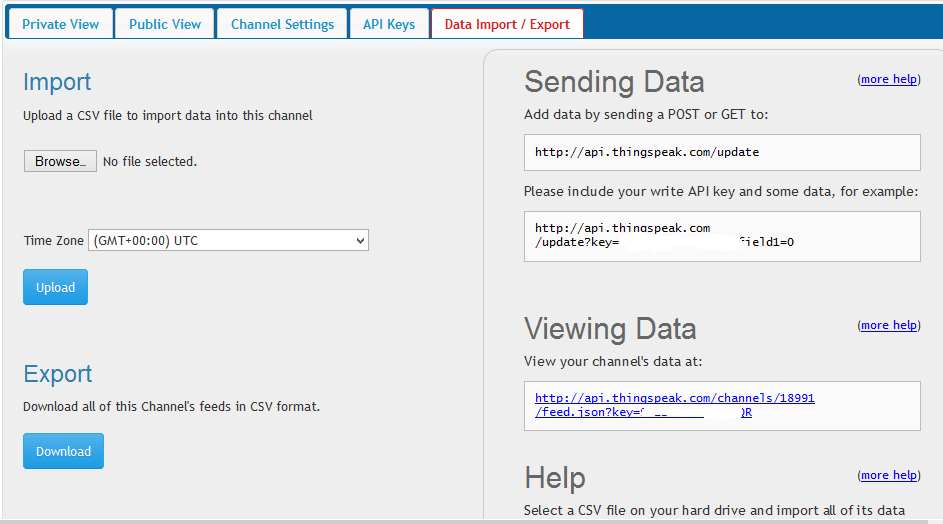
When we create a channel, by default, a write API key is generated. We generate read API keys by clicking the ‘Generate New Read API Key’ button under this tab.You can also add a note corresponding to each of the read API keys.

**Note:** Please note that clicking on the ‘Generate New Write API Key’ will over-write the previous key. You will only have one Write API key at any point of time. Also, in case your channel is private, others can only view the channel’s feed and charts by using a Read

API key. Please share the Read API keys with people who are approved and authorized to view your channel.

Now click on the ‘Data Import/Export’ tab and you should see a screen similar to the below. This tab is used to import the ‘Comma Separated Values(CSV)’ data from a file into the channel. You

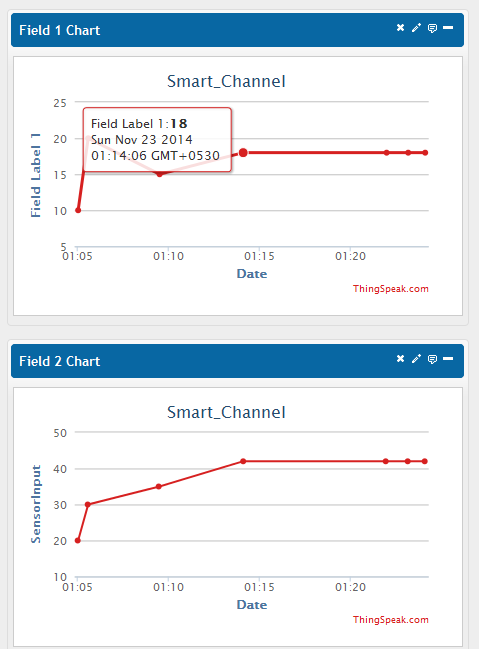
can also download the channel’s feed from here in CSV format. This tab also outlines how to send and view data by providing the URIs to the send and view APIs.



**Exploring the code:** I have started with building the URI for updating the fields. The initial URI is<http://api.thingspeak.com/update>to which the key has to be appended. Please note that you need to replace the string YOUR\_KEY with the actual write API key which you will get once you sign up and create a channel.

After that I have declared two constants which I will be pushing to my ThingSpeak channel. After that I am appending these fields to the URL under the parameters field1 and field2. You can push the other fields until field8 similarly. After this, I am creating a HTTPWebRequest object corresponding to this URI by using the ‘WebRequest.Create’ method. After this, I am retrieving the response using the GetResponse method and assigning the same to the response object. In case the data was successfully sent to the channel, the resulting StatusDescription will be ‘OK’. I am checking for any non-OK status descriptions and just in case of an error, I am throwing an exception and displaying the resultant message in a label.

After a series of updates, the charts in the private view tab for each of the fields will look like the below:



Each of the dots correspond to the value and the time at which the value was posted to the channel. Place the mouse over a dot to get more details on the exact date and the GMT offset from which the value was posted.

Please note that in the above example, I have sent some sample values to the channel. You can send any data here, say the periodic readings from a temperature sensor or RPM values from a motor. The Y-axis show the names that we specified to each of the labels.

**ThingSpeak Apps**

ThingSpeak provides apps that allow us for an easier integration with the web services, social networks and other APIs. Below are some of the apps provided by ThingSpeak:

* ThingTweet - This allows you to post messages to twitter via ThingSpeak. In essence, this is a TwitterProxy which re-directs your posts to twitter.
* ThingHTTP - This allows you to connect to web services and supports GET, PUT, POST and DELETE methods of HTTP.
* TweetControl - Using this, you can monitor your Twitter feeds for a specific key word and then process the request. Once the specific keyword is found in the twitter feed, you can then use ThingHTTP to connect to a different web service or execute a specific action.
* React - Send a tweet or trigger a ThingHTTP request when the Channel meets a certain condition.

**CHAPTER 8**

# **CONCLUSION**

In conclusion, the solar-powered, temperature-controlled e-uniform offers a groundbreaking solution to the challenges soldiers face in extreme and unpredictable weather conditions. By integrating solar power, temperature sensors, microcontrollers, LED heating elements, and DC motors, the uniform ensures soldiers maintain optimal body temperature in both hot and cold environments, preventing heat exhaustion and hypothermia. Powered by solar energy, it is self-sustaining, making it ideal for use in remote or energy-scarce areas.

Additionally, the inclusion of GPS tracking enhances situational awareness, improving safety and mission coordination. The uniform is lightweight, durable, and comfortable, allowing for enhanced mobility without sacrificing protection. Its eco-friendly design reduces environmental impact by minimizing the need for external power sources. Overall, the e-uniform improves soldier well-being, operational efficiency, and safety, offering a comprehensive and sustainable solution to the challenges posed by extreme climates.

This project represents a significant step forward in military uniform innovation.

This uniform not only improves thermal regulation but also reduces the physiological burden on soldiers, enhancing their endurance and focus during critical missions. The integration of lightweight materials and advanced wearable technology further underscores its practicality and usability in real-world scenarios.

Ultimately, the Smart Solar-Driven E-Uniform aligns with the modern military's goals of sustainability, innovation, and enhanced soldier welfare, offering a promising solution for the challenges posed by diverse and unpredictable climates.

**REFERENCES**

* Vignesh. A; C. Bhuvaneswari; Shiva M; W. Abitha Memala; M. Pushpavalli; M. Kavitha, Intelligent Solar Based Climate Adjustable E-Uniform for Soldiers, 2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI).
* [Jeeva B](https://ieeexplore.ieee.org/author/37085355287); [Bangi Karuna Sree](https://ieeexplore.ieee.org/author/287500453812907); [Basineni Udaya Priyanka](https://ieeexplore.ieee.org/author/264913915911946); [Divyadarshini S](https://ieeexplore.ieee.org/author/402993462191312); [Hema. S](https://ieeexplore.ieee.org/author/904069591585913), Smart E-uniform for Soldiers, [2023 7th International Conference on Design Innovation for 3 Cs Compute Communicate Control (ICDI3C)](https://ieeexplore.ieee.org/xpl/conhome/10592106/proceeding)
* [Chowdam Anila](https://ieeexplore.ieee.org/author/915308729493788); [Ashokkumar. N](https://ieeexplore.ieee.org/author/37089828460); [Eduru Joel Vittan](https://ieeexplore.ieee.org/author/383146783513101); [Bheemaneni Uma Maheswara Chowdary](https://ieeexplore.ieee.org/author/388068685652523); [Dhanya Sree Bala Subramani](https://ieeexplore.ieee.org/author/37086245031); [Balamanikandan. A](https://ieeexplore.ieee.org/author/159289354601205), [Sun-Powered Tactical Uniforms for Soldiers in Military Applications](https://ieeexplore.ieee.org/document/10717124/), [2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS)](https://ieeexplore.ieee.org/xpl/conhome/10716817/proceeding)
* Rahul Khairamode, Rahul Khangouka and Rutik Patil, "Design And Fabrication Of Solar Based E-Jacket For Soldiers", International Journal of Engineering Technology and Management Sciences, July 2022.
* Dipali H. Kale, Yogita P. Phapale, S. Kamble Shivani and S. L. Shirke, "Solar Based E-Uniform for Soldiers", International Research Journal of Engineering and Technology (IRJET), May 2021.
* A. Abi Zeid Daou, C. Haddad and R. Abi Zeid Daou, "Design and Implementation of a Smart Soldier Uniform", 2021 IEEE 3rd International Multidisciplinary Conference on Engineering Technology (IMCET), pp. 130135, 2021.
* S.M.D. Tuljapurkar, Ashitosh Gadhve, Sumit Gulve, Nilesh Madane and Ajit Kolpe, "Solar Based E-Uniform For Soldiers Working at Extreme Weather Conditions", Solar Based E-Uniform For Soldiers Working at Extreme Weather Conditions, June 2019
* S. Ramesh, Dr. R. Sankarganesh, N. Dharani and P. Kandeepan, "Climate Adjustable E- Military Suit", International Research Journal of Modernization in Engineering Technology and Science, March 2020.
* M. Vasanthi, S. Anitha, M. Suganya, S. Ashma, K. Janani and M. Kaviyasri, E-Smart Army Jacket Based On Climate Condition Using Arm Microcontroller, vol. 9, no. 5, 2020.

A. Abi Zeid Daou, C. Haddad and R. Abi Zeid Daou, "Design and Implementation of a Smart Soldier Uniform", *2021 IEEE 3rd International Multidisciplinary Conference on Engineering Technology (IMCET)*, pp. 130135, 2021.