A

PRACTICAL FILE

ON

Internet of Things (IOT) (CS-801)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD

OF THE DEGREE

BACHELOR OF TECHNOLOGY

(Computer Science & Engineering)

SUBMITTED TO

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

SUBMITTED TO:

Mr. Vineet Raj Singh Kushwaha

Associate Professor

Department of Computer Science

& Engineering

SUBMITTED BY:

DEEPAK SINGH TOMAR

ROLL NO. - 0928CS201028

B.Tech CSE 4th year

8th semester

Department Of Computer Science & Engineering

IPS COLLEGE OF TECHNOLOGY AND MANAGEMENT, GWALIOR

May-2024

Content

Experiment – 1 **Study and Install IDE of Arduino and different types of Arduino.**

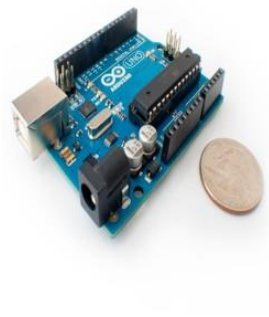
Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

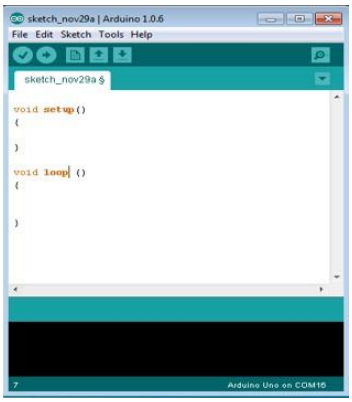
Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are −

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.





Board Types

Various kinds of Arduino boards are available depending on different microcontrollers used. However, all Arduino boards have one thing in common: they are programed through the Arduino IDE.

The differences are based on the number of inputs and outputs (the number of sensors, LEDs, and buttons you can use on a single board), speed, operating voltage, form factor etc. Some boards are designed to be embedded and have no programming interface (hardware), which you would need to buy separately. Some can run directly from a 3.7V battery, others need at least 5V.

Here is a list of different Arduino boards available.

Arduino boards based on ATMEGA328 microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Uno R3	5V	16MHz	14	6	6	1	USB via ATMega16U2
Arduino Uno R3 SMD	5V	16MHz	14	6	6	1	USB via ATMega16U2
Red Board	5V	16MHz	14	6	6	1	USB via FTDI
Arduino Pro 3.3v/8 MHz	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
Arduino Pro 5V/16MHz	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino mini 05	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro mini 3.3v/8mhz	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro mini 5v/16mhz	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Ethernet	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino Fio	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header

LilyPad Arduino 328 main board	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
LilyPad Arduino simple board	3.3V	8MHz	9	4	5	0	FTDI-Compatible Header

Arduino boards based on ATMEGA32u4 microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Leonardo	5V	16MHz	20	12	7	1	Native USB
Pro micro 5V/16MHz	5V	16MHz	14	6	6	1	Native USB
Pro micro 3.3V/8MHz	5V	16MHz	14	6	6	1	Native USB
LilyPad Arduino USB	3.3V	8MHz	14	6	6	1	Native USB

Arduino boards based on ATMEGA2560 microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Mega 2560 R3	5V	16MHz	54	16	14	4	USB via ATMega16U2B
Mega Pro 3.3V	3.3V	8MHz	54	16	14	4	FTDI-Compatible Header
Mega Pro 5V	5V	16MHz	54	16	14	4	FTDI-Compatible Header
Mega Pro Mini 3.3V	3.3V	8MHz	54	16	14	4	FTDI-Compatible Header

Arduino boards based on AT91SAM3X8E microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Mega 2560 R3	3.3V	84MHz	54	12	12	4	USB native

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

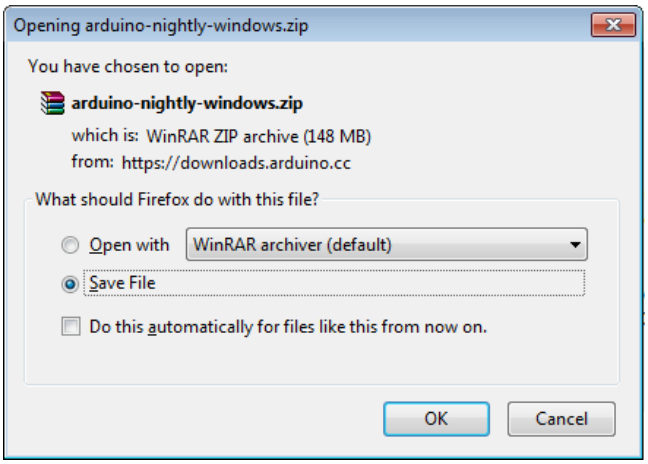
In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

Step 1 − First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.

Step 2 − Download Arduino IDE Software.

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.



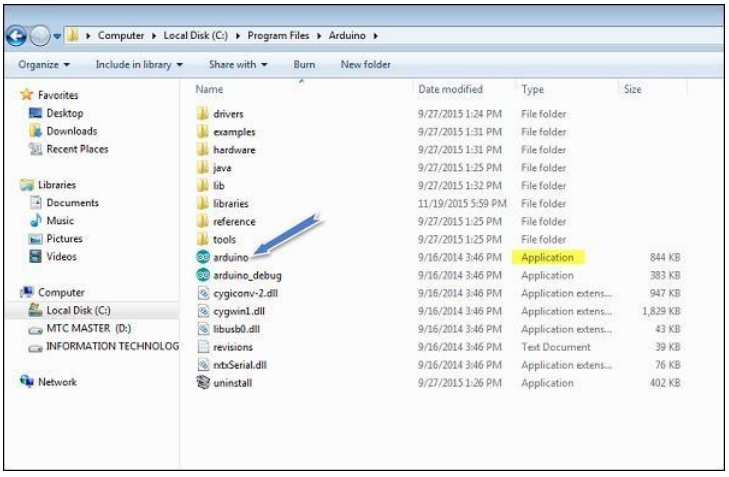
Step 3 − Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

Step 4 − Launch Arduino IDE.

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.

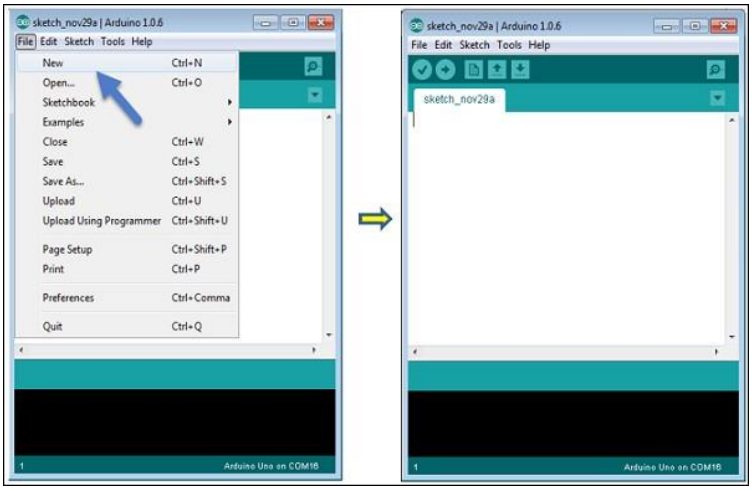


Step 5 − Open your first project.

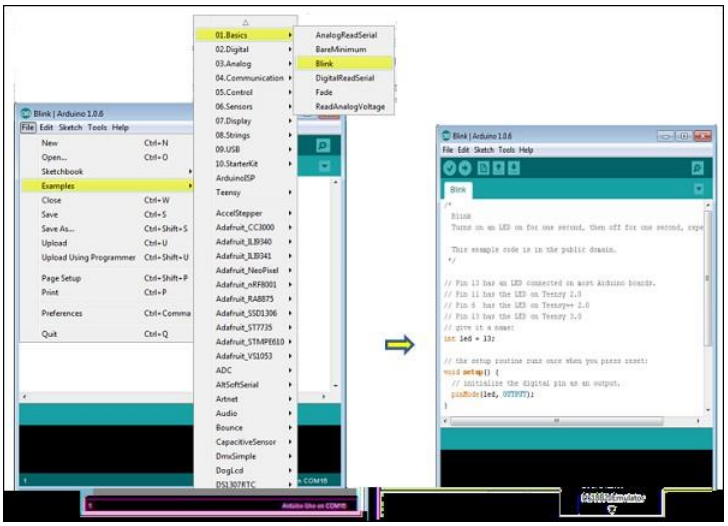
Once the software starts, you have two options −

- Create a new project.
- Open an existing project example.

To create a new project, select File → **New**.



To open an existing project example, select File → Example → Basics → Blink.

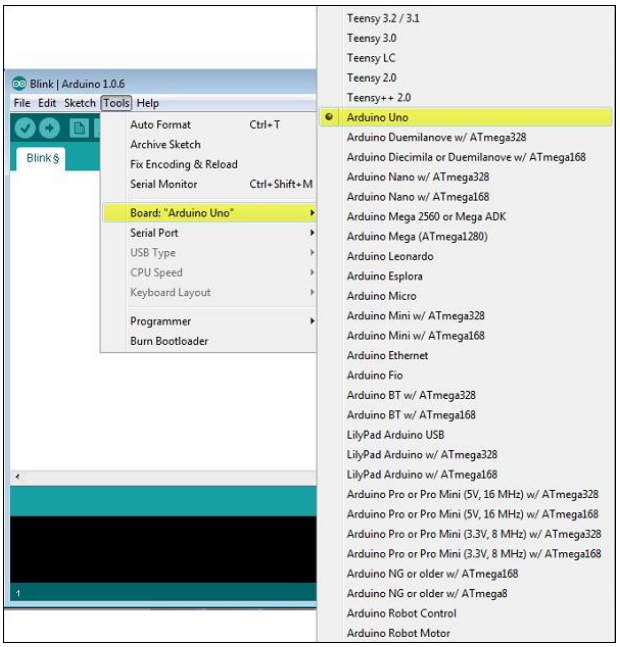


Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6 − Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

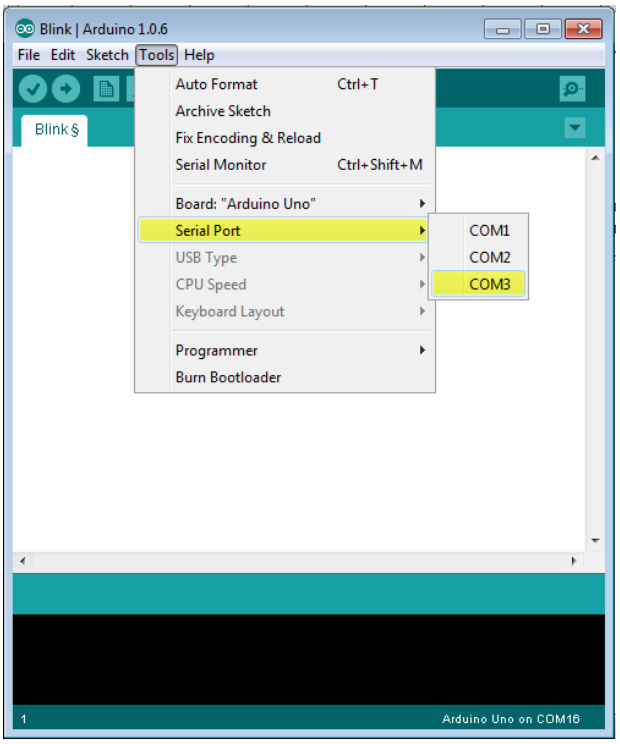
Go to Tools → Board and select your board.



Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using.

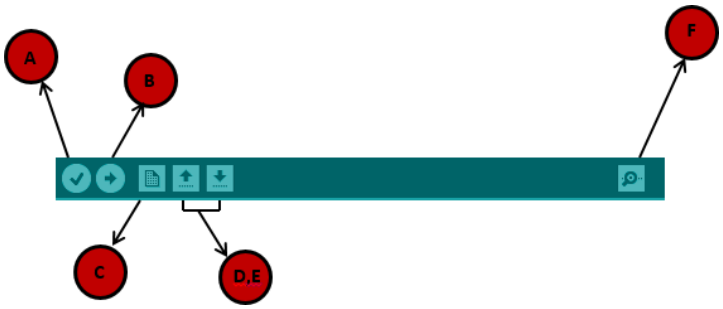
Step 7 − Select your serial port.

Select the serial device of the Arduino board. Go to **Tools → Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 8 − Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A − Used to check if there is any compilation error.

B − Used to upload a program to the Arduino board.

C − Shortcut used to create a new sketch.

D − Used to directly open one of the example sketch.

E − Used to save your sketch.

F − Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

Note − If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

Experiment – 2 **Write program using Arduino IDE to Blink LED.**

Blinking an LED

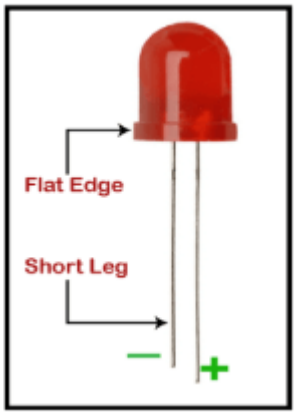
It is the simple basic project created using Arduino. LED (Light Emitting Diode) is an electronic device, which emits light when the current passes through its terminals. LED's are used in various applications. It is also used as an ON/OFF indicator in different electronic devices.

In this project, we will connect the LED to the digital pin on the Arduino board. The LED will work as a simple light that can be turned ON and OFF for a specified duration.

Structure of LED

An LED is a two-terminal device. The two terminals are called as Cathode and Anode.

It is shown below:



The long terminal is called Anode, and the shorter terminal is called Cathode. Here, cathode is the negative terminal and anode is the positive terminal.

Components of the project

The components used in the blinking of an LED are listed below:

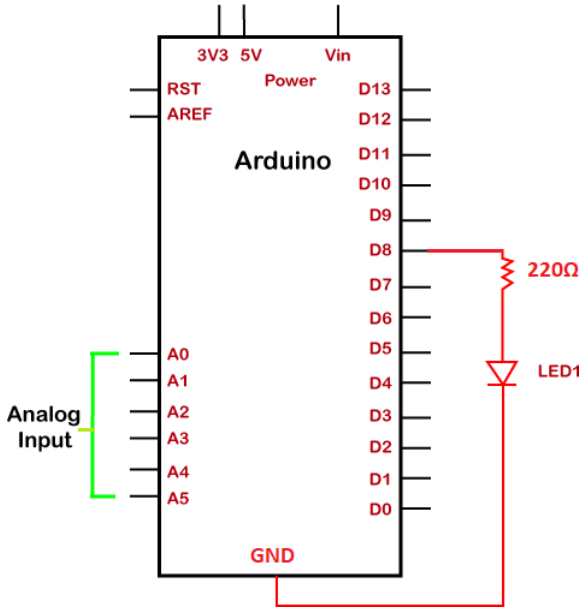
1. 1 x Arduino UNO board.
   We can use any version of the UNO board, such as UNO R3, etc. We can also use other types of Arduino boards, such as Arduino Zero, Arduino Micro, etc.
2. 1 x Breadboard
3. 2 x Jump wires
4. 1 x LED
5. 1 x Resistor of 220 Ohm.

We can use a resistor of any value upto 470 Ohms. We can use other value of resistors as well, depending on our circuit requirements. Usually, the value should not exceed the allowable forward current.

Structure of the project

The structure clearly shows the pinout of the UNO board. It also displays the LED and resistance connected to the board.

It is shown below:



Sketch

We need to install the Arduino IDE, to begin with the coding, which is already discussed.

Open the IDE and start with the coding, which is given below:

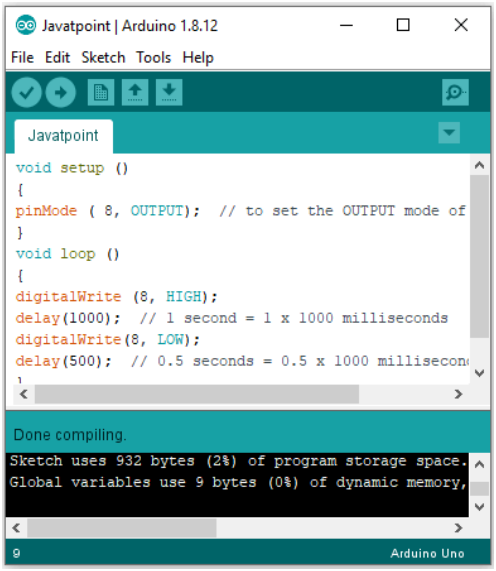
```
1. void setup ()
2. {
3. pinMode ( 8, OUTPUT);  // to set the OUTPUT mode of pin number 8.
4. }
5. void loop ()
6. {
7.   digitalWrite (8, HIGH);
8.   delay(1000);  // 1 second = 1 x 1000 milliseconds
9.   digitalWrite (8, LOW);
10. delay(500);  // 0.5 second = 0.5 x 1000 milliseconds
11. }
```

We can modify the delay duration according to our choice or as per the requirements.

Every statement of coding is explained in Arduino coding basics. You can open the URL for clear understanding.

Note: Make sure the code is free of errors.

The sketch will be uploaded to the board after the correct compiling, as shown below:



We are required to click on the Verify button to compile the code.

The RX and TX LED on the board will light up after the successful uploading of the code.

Experiment – 3 **Write Program for RGB LED using Arduino.**

The RGB LED can emit different colors by mixing the 3 basic colors red, green and blue. So it actually consists of 3 separate LEDs red, green and blue packed in a single case. That's why it has 4 leads, one lead for each of the 3 colors and one common cathode or anode depending of the RGB LED type. In this tutorial I will be using a common cathode one.

Components needed for this tutorial

Arduino and RGB LED Circuit Schematics

The cathode will be connected to the ground and the 3 anodes will be connected through 220 Ohms resistors to 3 digital pins on the Arduino Board that can provide PWM signal. We will use PWM for simulating analog output which will provide different voltage levels to the LEDs so we can get the desired colors.

We will use PWM for simulating analog output which will provide different voltage levels to the LEDs so we can get the desired colors.

Source Code

Now let's see the Arduino sketch. I will use the pins number 7, 6 and 5 and I will name them redPin, greenPin and bluePin. In the setup section we need to define them as outputs. At the bottom of the sketch we have this custom made function named setColor() which takes 3 different arguments redValue, greenValue and blueValue. These arguments represents the brightness of the LEDs or the duty cycle of the PWM signal which is created using the analogWrite() function. These values can vary from 0 to 255 which represents 100 % duty cycle of the PWM signal or maximum LED brightness.

```
int redPin= 7;

int greenPin = 6;

int bluePin = 5;

void setup() {

  pinMode(redPin, OUTPUT);

  pinMode(greenPin, OUTPUT);

  pinMode(bluePin, OUTPUT);
```

```
}

void loop() {

  setColor(255, 0, 0); // Red Color

  delay(1000);

  setColor(0, 255, 0); // Green Color

  delay(1000);

  setColor(0, 0, 255); // Blue Color

  delay(1000);

  setColor(255, 255, 255); // White Color

  delay(1000);

  setColor(170, 0, 255); // Purple Color

  delay(1000);

}

void setColor(int redValue, int greenValue, int blueValue) {

  analogWrite(redPin, redValue);

  analogWrite(greenPin, greenValue);

  analogWrite(bluePin, blueValue);

}
```

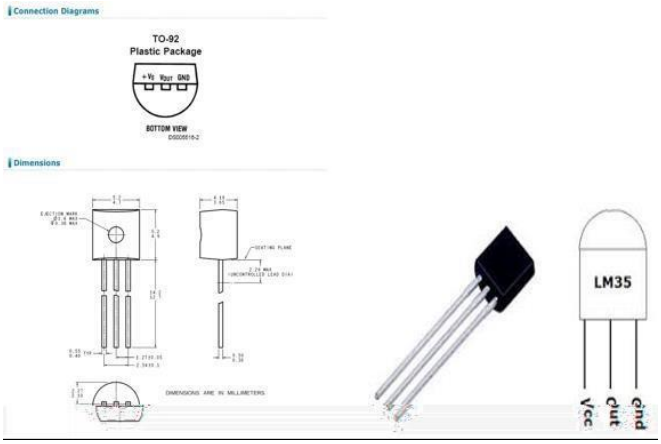
Code language: Arduino (arduino)

So now in the loop function we will make our program which will change the color of the LED each second. In order to get red light on the LED we will call the setColor() function and set value of 255 for the redValue argument and 0 for the two others. Respectively we can get the two other basic colors, green and blue. For getting other colors we need to mix the arguments values. So for example if set all 3 LEDS to maximum brightness we will get White color and we will get a purple color if we set the following values to the arguments: 170 redValue, 0 greenValue and 255 blueValue. Here's the demonstration of the sketch.

Experiment - 4 **Study the Temperature sensor and Write a program to monitor temperature using Arduino.**

The Temperature Sensor LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature.

The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range.



Technical Specifications

- Calibrated directly in Celsius (Centigrade)
- Linear + 10-mV/°C scale factor
- 0.5°C ensured accuracy (at 25°C)
- Rated for full −55°C to 150°C range
- Suitable for remote applications

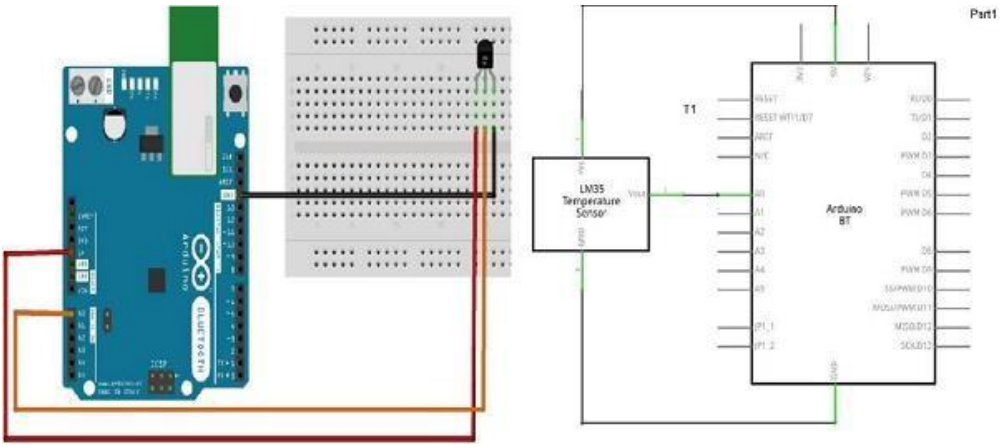
Components Required

You will need the following components −

- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × LM35 sensor

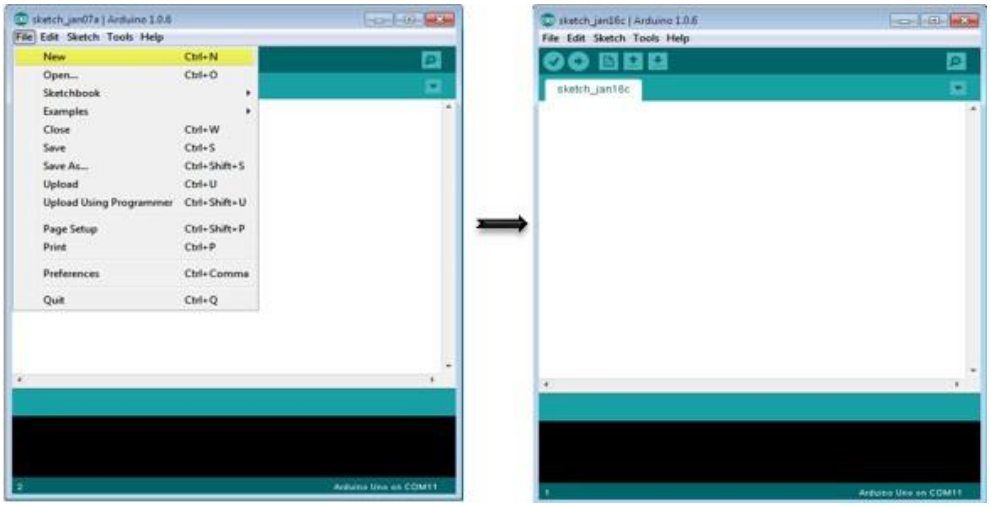
Procedure

Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below.



Sketch

Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit. Open a new sketch File by clicking New.



Arduino Code

```
float temp;
int tempPin = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  temp = analogRead(tempPin);
  // read analog volt from sensor and save to variable temp
  temp = temp * 0.48828125;
  // convert the analog volt to its temperature equivalent
  Serial.print("TEMPERATURE = ");
  Serial.print(temp); // display temperature value
```

```
   Serial.print("*C");
   Serial.println();
   delay(1000); // update sensor reading each one second
}
```

Code to Note

LM35 sensor has three terminals - V_s, V_{out} and GND. We will connect the sensor as follows −

- Connect the $+V_s$ to +5v on your Arduino board.
- Connect V_{out} to Analog0 or A0 on Arduino board.
- Connect GND with GND on Arduino.

The Analog to Digital Converter (ADC) converts analog values into a digital approximation based on the formula ADC Value = sample * 1024 / reference voltage (+5v). So with a +5 volt reference, the digital approximation will be equal to input voltage * 205.

Result

You will see the temperature display on the serial port monitor which is updated every second.

Experiment - 5 **Write a program to study Bluetooth connectivity with NodeMCU.**

HC-05 is a Bluetooth device used for wireless communication with Bluetooth-enabled devices (like smartphones). It communicates with microcontrollers using serial communication (USART).

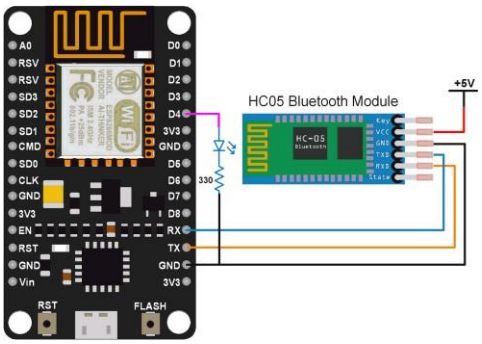
This device adds wireless communication protocol to the embedded applications through which it can communicate with any other Bluetooth enabled device.

AT commands are used to command the Bluetooth module. We can change its settings about the password, its name, USART communications settings like baud rate, no. of stop bits or parity, etc.

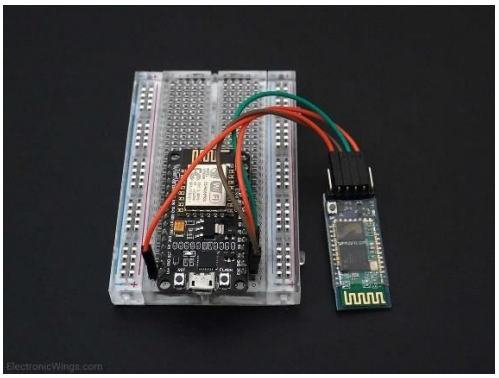
For more information about the HC-05 Bluetooth module, its pin, and how to use it, refer to the topic **HC-05 Bluetooth** module in the sensors and modules section.

NodeMCU based ESP8266 has UART serial communication module through which it can communicate Bluetooth module. to know about Lua-based NodeMCU UART functions refer to **NodeMCU UART with ESPlorer IDE**.

Connection Diagram of HC-05 Bluetooth Module with NodeMCU



HC-05 Bluetooth module interface with NodeMCU



Note: Default Bluetooth name of the device is “HC-05” and the default PIN (password) for connection is either “0000” or “1234”.

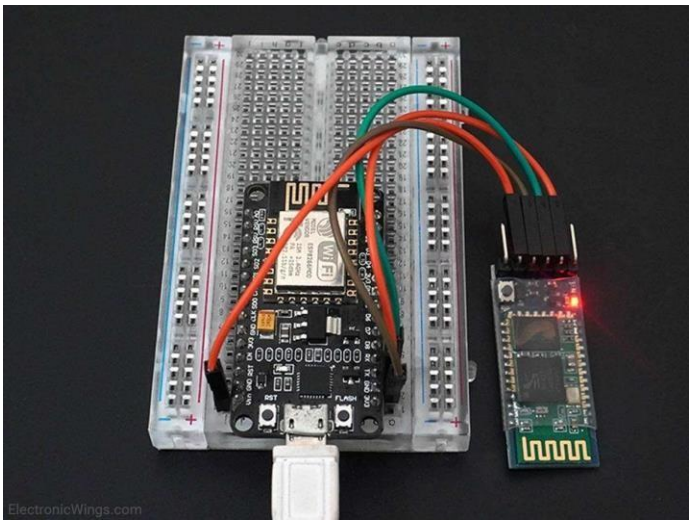
Control LED using HC-05 Bluetooth Module using NodeMCU

Let’s develop a small application in which we can control LED ON-OFF through a smartphone.

This is done by interfacing the HC-05 Bluetooth module with NodeMCU. Data from HC-05 is received/ transmitted serially by NodeMCU.

In this application, when 1 is sent from the smartphone, LED will turn ON and if 2 is sent LED will get Turned OFF. If received data is other than 1 or 2, it will return a message to mobile that select the proper option.

We can write codes for NodeMCU DevKit in either Lua Script or C/C++ language. We are using ESPlorer IDE for writing code in Lua scripts and Arduino IDE for writing code in C/C++. To know more refer to Getting started with NodeMCU using ESPlorer IDE (which uses Lua scripting for NodeMCU) and Getting started with NodeMCU using Arduino IDE (which uses C language-based Arduino sketches for NodeMCU).

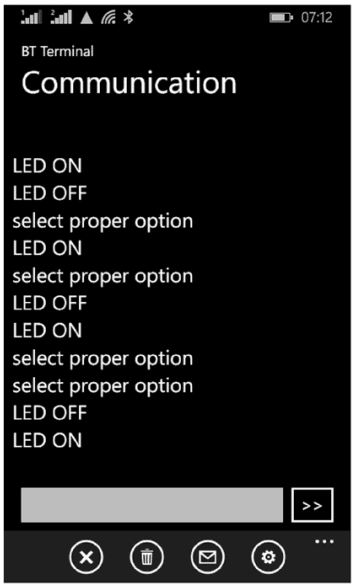


Lua Script for Bluetooth

```
LEDpin = 4


gpio.mode(LEDpin, gpio.OUTPUT)--set LED pin as output pin
gpio.write(LEDpin, gpio.LOW)-- set LED state initially low
```

```
--begin uart with specs
uart.setup(0, 9600, 8, uart.PARITY_NONE, uart.STOPBITS_1, 1)
--set callback function on receive to make decision about LED on/off
uart.on("data",1,
function(data)
   if(data == "1") then
      gpio.write(LEDpin, gpio.HIGH)
      print("LED ON")
   elseif(data == "2") then
      gpio.write(LEDpin, gpio.LOW)
      print("LED OFF")
   else
      print("select proper option")
   end
end, 0)
```

Below is the response received from NodeMCU Bluetooth while sending commands of the above example on the Bluetooth terminal of the smartphone.



Also, we can write code for the above example from Arduino IDE. To know about how to start using NodeMCU with Arduino IDE refer to Getting started with NodeMCU using Arduino IDE.

HC-05 Bluetooth Code for NodeMCU using Arduino

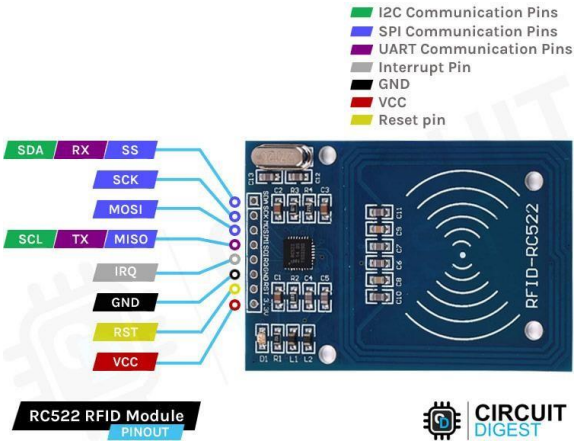
```
int LED = D4;


void setup() {
 pinMode(LED, OUTPUT);
 Serial.begin(9600); /* Define baud rate for serial communication */
}


void loop() {


 if (Serial.available()) /* If data is available on serial port */
   {
     char data_received;
     data_received = Serial.read();  /* Data received from bluetooth */
     if (data_received == '1')
     {
      digitalWrite(LED, HIGH);
      Serial.write("LED turned ON\n");
     }
     else if (data_received == '2')
     {
      digitalWrite(LED, LOW);
      Serial.write("LED turned OFF\n");
     }
     else
```

```
    {
     Serial.write("Select either 1 or 2");
    }
   }
}
```

Experiment - 6 **Study and Implement RFID using Arduino.**

RC522 RFID Reader/Writer Module Pinout

The RC522 module has a total of 8 pins. This module supports various communication protocols and each pin has a different function for each communication protocol. The **pinout of a RFID Reader module** is as follows:



SDA SCL I2C Communication pins. DATA and CLOCK.

SS SCK MOSI MISO SPI communication pins. Slave Select, Clock, MOSI, and MISO.

RX TX UART Communication pins.

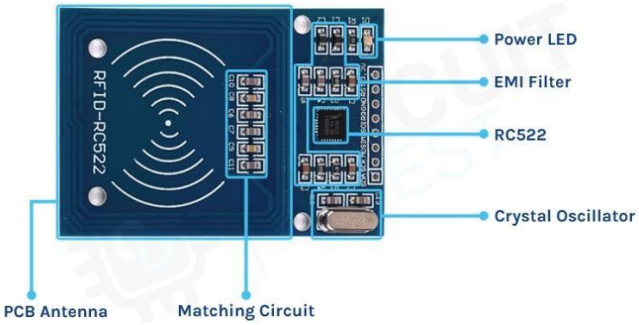
IRQ Interrupt signal from the module to indicate RFID tag detection.

GND Ground pin that needs to be connected to the GND pin on the Arduino.

RST Reset pin for the module

VCC Supply pin for the module. The supply voltage can be anywhere from 2.5V to 3.3V and must be connected to the 3.3V pin on the Arduino.

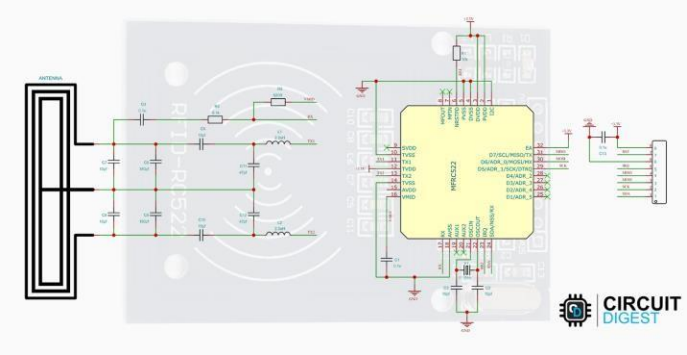
RC522 RFID Reader/Writer Module - Parts



The RC522 module consists of an **MFRC522 RFID chip** from NXP. It is clocked at 27.12MHz by the onboard crystal oscillator in the 49S package. The board also consists of the required EMI filter circuit and matching circuit. It also consists of a PCB antenna for communicating and energizing the RFID tags.

RC522 RFID Reader/Writer Module Circuit Diagram

The **Schematic diagram for the RC522 module** is given below. The circuit consists of bare minimum components. As we know the main component is the MFRC522 chip. The remaining components form the EMI filter along with the matching circuit for the antenna.



How is the RFID Works?

The RFID system is comprised of two components: the RFID reader and the tags. They are also called PCD (Proximity Coupling Device) and PICC (Proximity Integrated Circuit Card).

The RFID reader consists of an antenna to emit high-frequency EM waves and a reader/writer. MFRC522 from NXP is an example of such an integrated circuit. Since we are using high-frequency waves in the megahertz range, the size of the antenna can be small.

The RFID tag can be either passive or active. Active tags are powered by batteries while the passive RFID tags are powered by energy from the reader's interrogating EM waves. The tags are available in different forms or shapes like cards, tags, key forbs, or stickers. Whatever the shape, the RFID tag will consist of an antenna and the RFID chip, which will store all the data. When triggered by an electromagnetic interrogation pulse from a nearby RFID reader, the tag will transmit data back to the reader. The reader will then analyze this data to identify the tag. Unlike a barcode or a QR code, the tag does not need to be within the reader's line of sight. This makes it easier to process and can be used for tracking objects in closed space.

Commonly Asked Questions about RFID Reader

What is an RFID used for?

RFID tags are a type of tracking system that uses radiofrequency to search, identify, track, and communicate with items and people.

How is RFID used in healthcare?

RFID helps to mitigate drug counterfeiting, simplifies the clinical trial process, improves the accuracy of patient identification, eases inventory management, streamlines patient tracking,

improves communications between caregivers and patients, and eradicates the risk of administering the wrong medications.

Can Wi-Fi interfere with RFID?

Interference from other radio-frequency (RF) emitting devices (RFI), such as other RFID readers and Wi-Fi access points, can negatively impact RFID system performance.

RC522 RFID Reader Module

The RC522 RFID module is based on the popular MFRC522 RFID reader chip from NXP. These modules are cheap and available from most online stores. MFRC522 is a highly integrated RFID reader/writer IC for contactless communication at 13.56 MHz. The MFRC522 reader supports ISO/IEC 14443 A/MIFARE and NTAG. The operating voltage of the RC522 module is 2.5V – 3.3V. Even though the maximum supply voltage is 3.3V the communication pins are 5V tolerant. So, we can connect the module directly to an Arduino without any Level-Shifters.

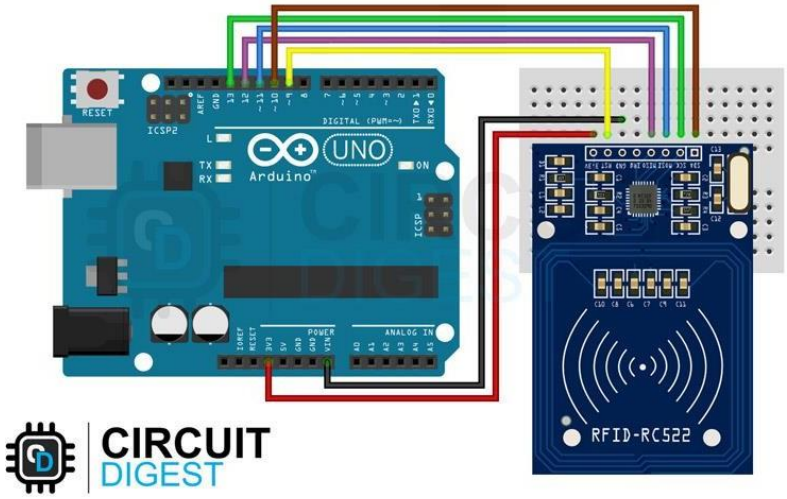
The MFRC522 supports three different communication protocols:

- **SPI** with Speed up to 10Mbit/s
- **I2C** interface with speed up to 400kBd in Fast mode and up to 3400kBd in High-Speed mode
- **RS232 Serial UART** with speed up to 1228.8kBd

The RC522 module usually comes with an RFID card and a key fob. And each of these comes with a 1KB of memory. We can not only read these tags but can also program these with the RC522 module. Here is the picture showing the tags along with the RC522 module.

Arduino RC522 RFID Reader Interfacing Circuit Diagram

For **interfacing the RC522 RFID module with the Arduino**, we will be using the SPI interface. Follow the circuit diagram and make the connections as per that.



The VCC and GND pins of the module are connected to the 3.3V and GND pins of Arduino respectively. The Reset pin is connected to the D9 and SS, MOSI, MISO, and SCK pins are connected to the D10, D11, D12, and D13 pins of the Arduino respectively. The SS and RST pins are configurable and can be connected to any other digital pins on the Arduino.

Arduino RC522 RFID Module Code

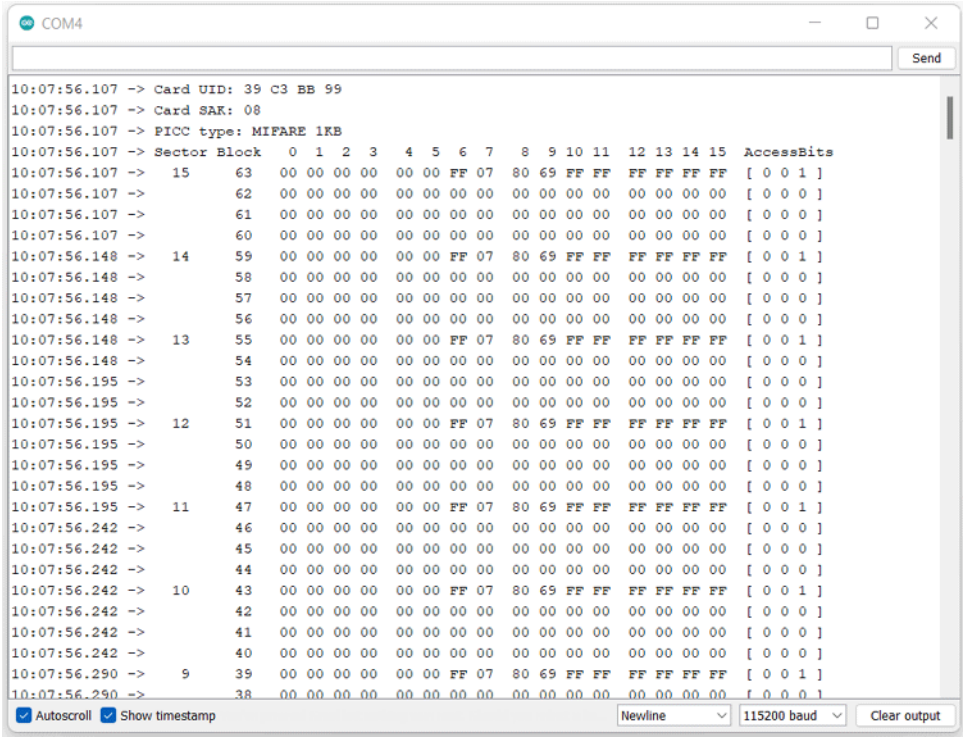
As the connections are made, let's look at the coding part. For that, we are going to use the **MFRC522 Arduino Library** by Miguel André Balboa. Since the library is not available in the Arduino library manager, download it from the MFRC522 GitHub repository and install it in the Arduino library folder. You can install it either through the Arduino IDE, by going to **Sketch -> Include Library -> Add ZIP Library** and selecting the downloaded **.ZIP** file, or by just simply extracting the Zip file into the Arduino library folder.

Once the library is installed, we can test our setup with an example code. For that, open the **DumpInfo** example from the MFRC522 library. Here is the example code.

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN         9          // Configurable, see typical pin layout above
#define SS_PIN          10         // Configurable, see typical pin layout above
MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance
void setup() {
Serial.begin(115200);                   // Initialize serial communications with the PC
while (!Serial);                    // Do nothing if no serial port is opened (added for Arduinos bas
ed on ATMEGA32U4)
```

```
        SPI.begin();                    // Init SPI bus
        mfrc522.PCD_Init();             // Init MFRC522
        delay(4);                       // Optional delay. Some board do need more time after init to be ready, see Readme
        mfrc522.PCD_DumpVersionToSerial();      // Show details of PCD - MFRC522 Card Reader details
        Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}
void loop() {
        // Reset the loop if no new card present on the sensor/reader. This saves the entire proc ess when idle.
        if ( ! mfrc522.PICC_IsNewCardPresent()) {
                return;
        }
        // Select one of the cards
        if ( ! mfrc522.PICC_ReadCardSerial()) {
                return;
        }
        // Dump debug info about the card; PICC_HaltA() is automatically called
        mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

Once it's compiled and uploaded to the Arduino, open the serial monitor and show any tag near the RC522 module. When the tag is in the vicinity, the reader will read all the data from the tag and will dump it into the serial monitor as shown below.



The data dump will contain all the details like Card UID, Card SAK, PICC type, and all the memory maps. The UID or Unique ID is Unique for each tag as the name suggests. If you get any communication failed error, that's because our serial baud rate is too slow. Increase the baud rate to 115200 in the code and it will resolve the issue. Here in the data dump, we can see that the PICC type is MIFARE 1KB. That means the tag contains a MIFARE chip with a memory of 1KB.

Experiment - 7 **Study and Configure Raspberry Pi.**

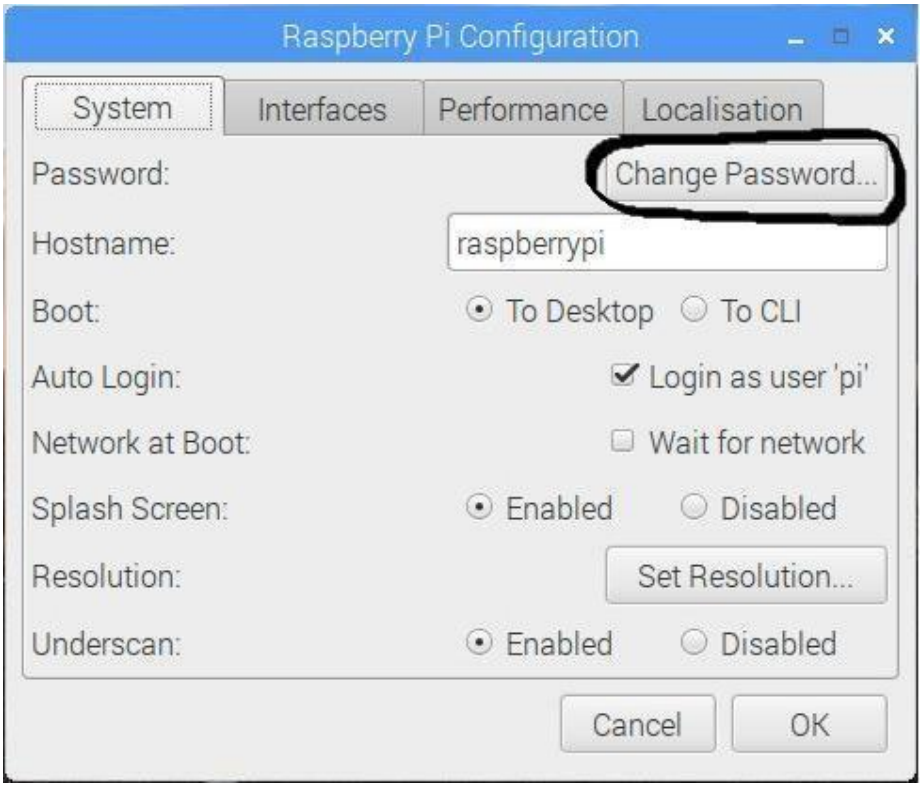
Raspbian configuration

For configuring Raspberry Pi in Raspbian, we are using Raspbian with PIXEL desktop. It is one of the best ways to get Raspbian started with the Raspberry Pi. Once we finish booting, we will be in the PIXEL desktop environment.

Now to open the menu, you need to click the button that has the Raspberry Pi logo on it. This button will be in the top left. After clicking the button, choose **Raspberry Pi configuration** from the preferences.

Configuration tool

Following is the configuration tool in PIXEL desktop −



By default, the configuration tool opens to its system tab which has the following options −

- **Change Password** − The default password is **raspberry**. You can change it by clicking the change password button.
- **Change the hostname** − The default name is **raspberry pi**. You can also change it to the name, which you want to use on the network.
- **Boot** − You can choose from the two options and control whether Raspberry Pi boots into the desktop or CLI i.e., command line interface.
- **Auto Login** − With the help of this option, you can set whether the user should automatically log in or not.

- **Network at Boot** − By choosing this option, you can set whether the pi user is automatically logged in or not.
- **Splash screen** − You can enable or disable it. On enabling, it will display the graphical splash screen that shows when Raspberry Pi is booting.
- **Resolution** − With the help of this option, you can configure the resolution of your screen.
- **Underscan** − There are two options, enable or disable. It is used to change the size of the displayed screen image to optimally fill the screen. If you see a black border around the screen, you should disable the underscan. Whereas, you should enable the underscan, if your desktop does not fit your screen.

There are three other tabs namely Interfaces, Performance, and Localization. The job of interface tab is to enable or disable various connection options on your Raspberry Pi.

You can enable the Pi camera from the interface tab. You can also set up a secure connection between computers by using SSH (short for Secure Shell) option.

If you want to remote access your Pi with a graphical interface then, you can enable RealVNC software from this tab. SPI, I2C, Serial, 1-wire, and Remote GPIO are some other interfaces you can use.

There is another tab called Performance, which will give you access to the options for overclocking and changing the GPU memory.

The localization tab, as the name implies, enable us to set −

- The character set used in our language.
- Our time zone.
- The keyboard setup as per our choice.
- Our Wi-Fi country.

Configure Wi-Fi

You can check at the top right, there would be icons for Bluetooth and Wi-Fi. The fan-shaped icon is on the Wi-Fi. To configure your Wi-Fi, you need to click on that icon. Once clicked, it will open a menu showing the available networks. It also shows the option to turn off your Wi-Fi.

Among those available networks, you need to select a network. After selecting, it will prompt for entering the Wi-Fi password i.e., the Pre Shared Key.

If you see a red cross on the icon, it means your connection has been failed or dropped. To test whether your Wi-Fi is working correctly, open a web browser and visit a web page.

Configure Bluetooth Devices

We can use wireless Bluetooth devices such as keyboard and/or mouse with Pi 3 and Pi zero W because these models are Bluetooth-enabled. In PIXEL desktop, you can set up your Bluetooth devices easily.

Following are the steps to configure the Bluetooth devices −

- First, make your device discoverable for pairing.
- Now, you need to click on the Bluetooth menu at the top right of the screen. It is aligned to the Wi-Fi button.
- Now, choose the Add Device option.
- The Raspberry will start searching for the devices and when it finds your device, click it and click the pair button.

Data Partition Setup

As we know that data partition is that area on your memory card (SD or MicroSD) which can be shared by various distributions. One of the best examples of use of a data partition is transferring the files between distributions.

The data partition has the **label** data.

You can use this labeled data to make a directory point to it as follows −

Step 1 − First, you need to boot the Raspberry Pi into Raspbian.

Step 2 − Now, click the Terminal icon to get to the command line.

Step 3 − Next, type the command **mkdir shared**. It will create a directory named **shared**.

Step 4 − Write the command **sudo mount -L data shared**. This command will point the directory to the shared partition.

Step 5 − Write the command **sudo chown $USER: shared**. It will set the permission for writing in this shared folder.

Step 6 − Now, to go to this shared folder, you need to type the command **cd shared**.

Once all the files are created in this shared folder, they will be available to all the distributions that have the permission to access the data partition.

Experiment - 8 **Write a program for LED blink using Raspberry Pi.**

The first step in this project is to design a simple LED circuit. Then we will make the LED circuit controllable from the Raspberry Pi by connecting the circuit to the general purpose input/output (GPIO) pins on the Raspberry Pi.

A simple LED circuit consists of a LED and resistor. The resistor is used to limit the current that is being drawn and is called a *current limiting resistor*. Without the resistor the LED would run at too high of a voltage, resulting in too much current being drawn which in turn would instantly burn the LED, and likely also the GPIO port on the Raspberry Pi.

To calculate the resistor value we need to examine the specifications of the LED. Specifically we need to find the forward voltage (VF) and the forward current (IF). A regular red LED has a forward voltage (VF) of 1.7V and forward current of 20mA (IF). Additionally we need to know the output voltage of the Raspberry Pi which is 3.3V.

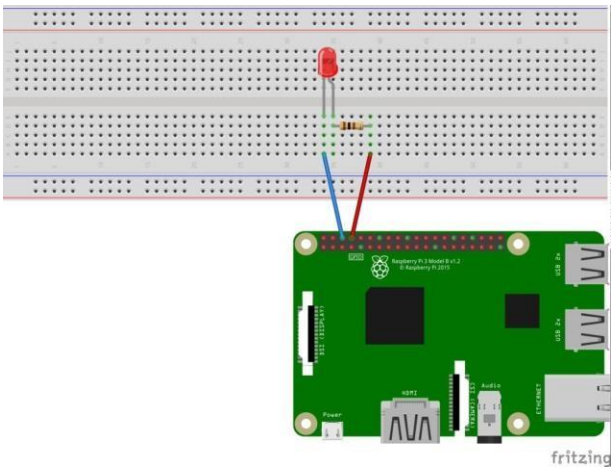
We can then calculate the resistor size needed to limit the current to the LED's maximum forward current (IF) using ohm's law like this:

$$R\Omega = VI = 3.3 - VFIF = 3.3 - 1.720mA = 80\Omega$$

Unfortunately 80 ohm is not a standard size of a resistor. To solve this we can either combine multiple resistors, or round up to a standard size. In this case we would round up to 100 ohm.

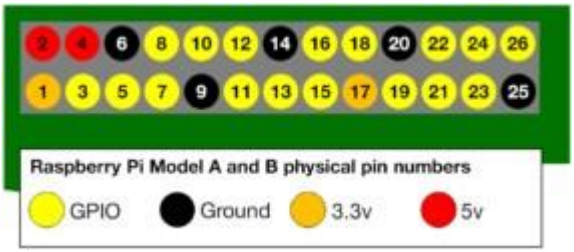
Important information: *Since ohm's law tells us that I (current) = V (voltage) / R (ohm) rounding up the resistor value will lower the actual current being drawn a little. This is good because we don't want to run our system at the maximum current rating. Rounding down instead of up would be dangerous, since it will actually increase the current being drawn. Increased current would result in running our system over the maximum rating and potentially destroying or damaging our components.*

With the value calculated for the current limiting resistor we can now hook the LED and resistor up to GPIO pin 8 on the Raspberry Pi. The resistor and LED needs to be in series like the diagram below. To find the right resistor use the resistor color code – for a 100 ohm resistor it needs to be brown-black-brown. You can use your multimeter to double check the resistor value.



When hooking up the circuit note the *polarity* of the LED. You will notice that the LED has a long and short lead. The long lead is the positive side also called the anode, the short lead is the negative side called the cathode. The long should be connected to the resistor and the short lead should be connected to ground via the blue jumper wire and pin 6 on the Raspberry Pi as shown on the diagram.

To find the pin number refer to this diagram showing the physical pin numbers on the Raspberry Pi.



Writing the Python Software to blink the LED

With the circuit created we need to write the Python script to blink the LED. Before we start writing the software we first need to install the Raspberry Pi GPIO Python module. This is a library that allows us to access the GPIO port directly from Python.

To install the Python library open a terminal and execute the following

```
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

With the library installed now open your favorite Python IDE (I recommend Thonny Python IDE more information about using it here).

Our script needs to do the following:

- Initialize the GPIO ports
- Turn the LED on and off in 1 second intervals

To initialize the GPIO ports on the Raspberry Pi we need to first import the Python library, the initialize the library and setup pin 8 as an output pin.

import RPi.GPIO **as** GPIO # Import Raspberry Pi GPIO library
from *time* **import** sleep # Import the sleep function from the time module
GPIO.setwarnings(**False**) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial value to low (off)

Next we need to turn the LED on and off in 1 second intervals by setting the output pin to either high (on) or low (off). We do this inside a infinite loop so our program keep executing until we manually stop it.

while True: # Run forever

GPIO.output(8, GPIO.HIGH) # Turn on
sleep(1) # Sleep for 1 second
GPIO.output(8, GPIO.LOW) # Turn off
sleep(1) # Sleep for 1 second

Combining the initialization and the blink code should give you the following full Python program:

import RPi.GPIO **as** GPIO # Import Raspberry Pi GPIO library
from *time* **import** sleep # Import the sleep function from the time module
GPIO.setwarnings(**False**) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial value to low (off)
while True: # Run forever
GPIO.output(8, GPIO.HIGH) # Turn on
sleep(1) # Sleep for 1 second
GPIO.output(8, GPIO.LOW) # Turn off
sleep(1) # Sleep for 1 second

With our program finished, save it as blinking_led.py and run it either inside your IDE or in the console with:

```
$ python blinking_led.py
```

With the program running you should see something like this:

You'll notice the program keeps on running because of the infinite loop. To stop it click either stop in your IDE or Ctrl+C if you run it inside the console.

Experiment - 9 **Study and Implement Zigbee Protocol using Arduino / Raspberry Pi**.

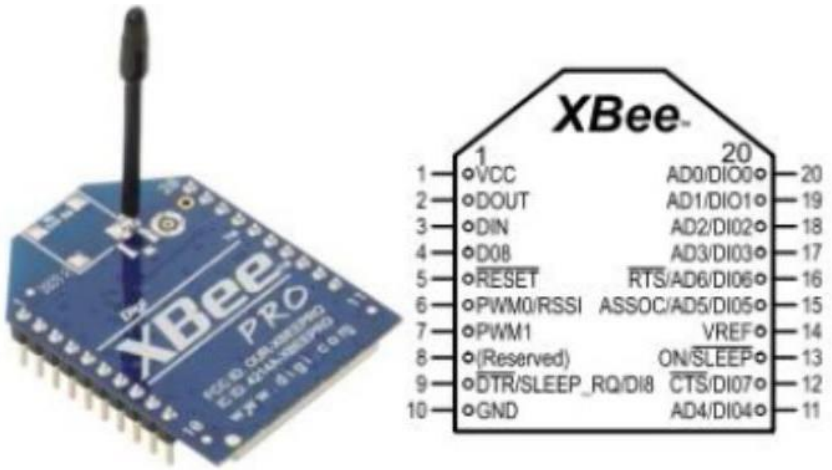
Zigbee is a wireless communication protocol targeted for battery powered devices (it has both low power and low cost). It generally operates in the 2.4GHz range (although there are geographic variations), and supports data ranges from 20 to 250 kbits/s.

The transmission distance though, is small compared to the likes of LoRa. It is 10 to 100 m, whereas LoRa can transmit over a few kilometers. Another thing to note is that Zigbee communication doesn't work very well if there is no line of sight between transmitter and receiver.

Even minor obstacles have been observed to significantly degrade the communication. Keep these limitations in mind when using Zigbee. You may want to look out for other options if your application can't meet these constraints.

In order to make Zigbee work with Arduino, we will use the XBee module.

These work with UART and therefore, it is fairly easy to interface them with Arduino. It is important to look at the pinout of XBee though, to understand which are the UART pins −



The DOUT and DIN pins in the figure above are the UART pins (TX and RX). They can be connected to two digital pins of Arduino (if you plan to use SoftwareSerial), or else to pins 0 and 1 of Arduino respectively (if you plan to use HW Serial). Please note that you won't be able to read print statements from the Arduino on the Serial Monitor if you use Hardware Serial for Zigbee interface.

Configuring the XBee modules

The XBee modules (transmitter and receiver) need to be configured using the X-CTU Software. It can be downloaded from **here**. This software is provided by DigiKey, and they have given a detailed configuration guide. Therefore, there is no point of me reinventing the wheel here. You can find the guide **here**.

There's another one by Sparkfun that is adapted to the newer version of the **X-CTU software**.

Here's another brief one by **Instructables**.

Please note that the two XBee modules that intend to communicate with each other should belong to the same series.

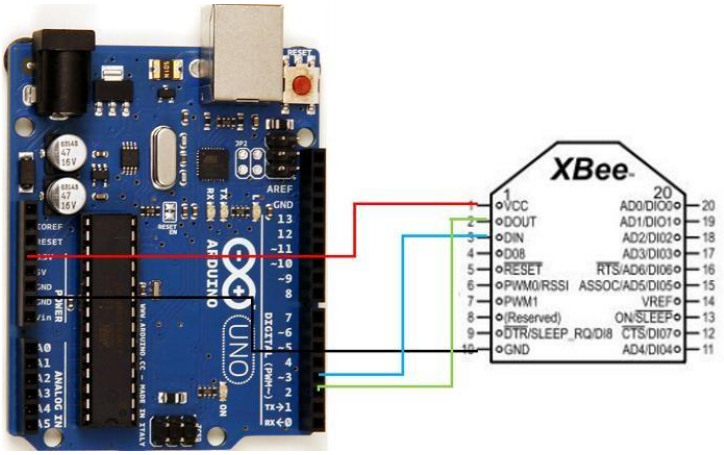
Here are a few things to note about the configuration −

- You will need a breakout board or an Explorer with a USB to UART converter for this configuration.
- The PAN ID (Personal Area Network ID) has to be the same for the devices that want to communicate with each other.
- One module needs to be set as the transmitter and the other as the receiver (this is determined by the CE field).

- Note the baud rate that you set. This will be used in the Arduino code, when configuring the Serial communication with XBee.

Circuit Diagram −

Once your XBee is configured, you can connect it to the Arduino via the breakout/Explorer board. In that case, the pinout will be slightly different depending on which board/ Explorer you are using. Here we will assume you are connecting the XBee directly to Arduino Uno, in which case, the connections will be −



As you can see, we have connected Vcc to 3.3V on Arduino, GND to GND, DOUT (TX) to pin 2, which will act as RX on the Arduino, and DIN (RX) to pin 3, which will act as TX on the Arduino.

The connections will be similar on the receiving side as well. If you have an on-board antenna, that's good, else you'll have to connect an antenna to the UFL connector.

Code Walkthrough

The code is quite straightforward. If you are using a board other than Arduino Uno, all digital pins may not support SoftwareSerial. Read the limitations of SoftwareSerial **here**

On the transmitting side, the code will be −

```
#include <SoftwareSerial.h>
SoftwareSerial xbeeSerial(2,3); //RX, TX

void setup() {
  Serial.begin(9600);
  xbeeSerial.begin(9600);
}
```

```
void loop() {
  if(Serial.available() > 0){
    char input = Serial.read();
    xbeeSerial.print(input);
  }
}
```

As you can see, whatever is sent by the user on the Serial Monitor is sent to the XBee module,and it will be received on the receiving side. The code for the receiving side is −

```
#include <SoftwareSerial.h>
SoftwareSerial xbeeSerial(2,3); //RX, TX

void setup() {
  Serial.begin(9600);
  xbeeSerial.begin(9600);
}

void loop() {
  if(xbeeSerial.available() > 0){
    char input = xbeeSerial.read();
    Serial.print(input);
  }
}
```

Over here, whatever is received from XBee is forwarded to the Serial Monitor. Thus, when testing out the combined system, whatever you type on the Serial Monitor on the transmitter side should be printed on the Serial Monitor on the receiver side.

Experiment - 10 **Study and implement MQTT protocol using Arduino.**

In this tutorial, we will create a setup that allows a Arduino UNO WiFi Rev2 board to send data to another Wi-Fi compatible board, using MQTT (Message Queuing Telemetry Transport). The sender device, simply publishes a message to a broker service, which then can be subscribed to by a receiver device.

The data we will send is simply random readings from the analog inputs on the Arduino UNO WiFi Rev2, but can easily be replaced by any sensor. This tutorial uses the broker **test.mosquitto.org**, an open-source service, free to use by anyone.

This tutorial uses the **ArduinoMqttClient** and **WiFiNINA** libraries.

Goals

The goals of this project are:

- Learn some basics of how MQTT works.
- Learn how to use the **ArduinoMqttClient** library.
- Create a sketch for a **publisher** device.
- Create a sketch for a **subscriber** device.

Hardware & Software Needed

- Arduino IDE (**online** or **offline**).
- **ArduinoMqttClient** library.
- **WiFiNINA** library.
- 2x Arduino UNO WiFi Rev2 (**link to store**).

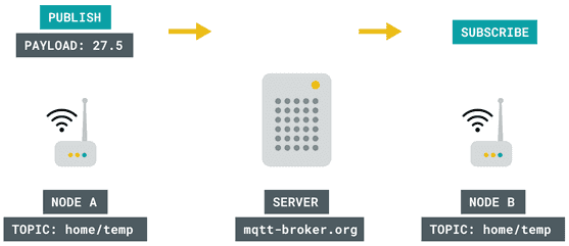
Note: The sketches in this tutorial also works with the MKR WiFi 1010 and Nano 33 IoT boards. You can for example use the Arduino UNO WiFi Rev2 as a publisher, and a Nano 33 IoT as a subscriber.

Message Queuing Telemetry Transport (MQTT)

The MQTT protocol was first introduced in 1999, as a light-weight **publish** and **subscribe** system. It is particularly useful for devices with low-

bandwidth, where we can send commands, sensor values or messages over the Internet with little effort.

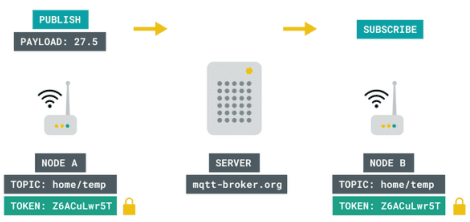
A basic explanation on how it works is that a node, for example and Arduino with a Wi-Fi module, sends a payload to a broker. A broker is a kind of "middle-point" server, that essentially stores payloads sent to it, in something called **topics**. A topic, is a definition of what type of data it contains, it could for example be "basement humidity" or "living room temperature". Another node can then subscribe to this information, from the broker, and voilà, data has been moved from Node A to Node B over the Internet.



Basics of MQTT.

There are several different ways this can be practiced, with many different layers of security depending on what type of broker and setup we use. For example, if we are dealing with non-sensitive data, such as temperature of a specific location, we are not too concerned on who might get access to it. But there's cases where data needs to be protected, for example in Social Media messaging services.

One way to protect the data is for example, by using a **token**, something that is quite common when working with various IoT services. For instance, if we are publishing something to a broker, anyone that has the URL, e.g. **randombroker.org/randomtopic** can subscribe to it. But if we add a unique token on both sides, they wouldn't be able to. These tokens could for example be **Z6ACuLwr5T**, which is not exactly something easy to guess.



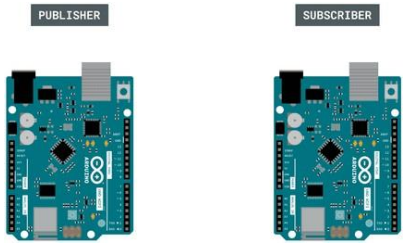
Encryption with MQTT.

MQTT is quite the interesting topic, and if you wish to read more about it, check out the links below:

- **mqtt.org** - all you need to know about MQTT.
- **Mosquitto.org** - an MQTT broker.
- **Inductive automation** - what is MQTT?
- **Randomnerdtutorials** - what is MQTT and how it works.

Circuit

This tutorial requires no external circuit. It does however, require two boards with a Wi-Fi module (one publisher and one subscriber).



The circuit.

Step by Step

We will now go through the steps required to setup one board as a publisher, and one as a subscriber. The following steps will be needed:

- Include the necessary libraries.
- Create a header file to store Wi-Fi credentials.
- Configure the **publisher device** to create three topics and publish them to a broker.
- Configure the **subscriber device** to subscribe to the three topics.

1. First, let's make sure we have the drivers installed. If we are using the Web Editor, we do not need to install anything. If we are using an offline editor, we need to install it manually. This can be done by navigating to **Tools > Board > Board Manager...**. Here we need to look for the **Arduino avrMEGA Boards** and install it.

2. Now, we need to install the libraries needed. If we are using the Web Editor, there is no need to install anything. If we are using an offline editor, simply go to **Tools >**

Manage libraries.., and search for **ArduinoMqttClient** and **WiFiNINA** and install them both.

3. Now let's take a look at some important functions used in the sketches:

- WiFiClient wifiClient

  - creates a Wi-Fi client.

- MqttClient mqttClient(wifiClient)

  - connects the Wi-Fi client to the MQTT client.

- WiFi.begin(ssid, pass)

  - connects to local Wi-Fi network.

- mqttClient.connect(broker, port)

  - connects to broker (and port).

- mqttClient.poll()

  - keeps the connection alive, used in the

  loop()

  .

- mqttClient.beginMessage(topic)

  - creates a new message to be published.

- mqttClient.print()

  - prints the content of message between the ().

- mqttClient.endMessage()

  - publishes the message to the broker.

- mqttClient.subscribe(topic)

  - subscribes to a topic.

- mqttClient.available()

  - checks if any messages are available from the topic.

- mqttClient.read()

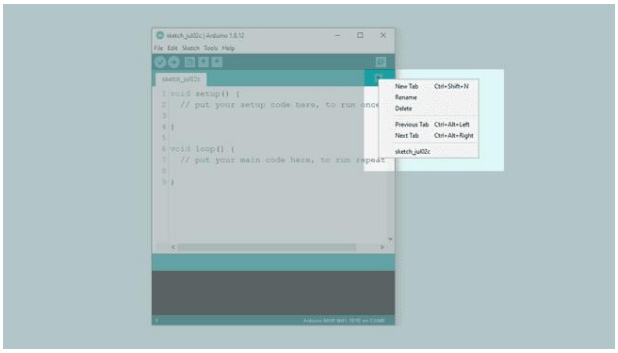
  - reads the incoming messages.

Programming the Publisher

4. We will now program the publisher device. Let's start by opening an empty sketch, and create a header file called

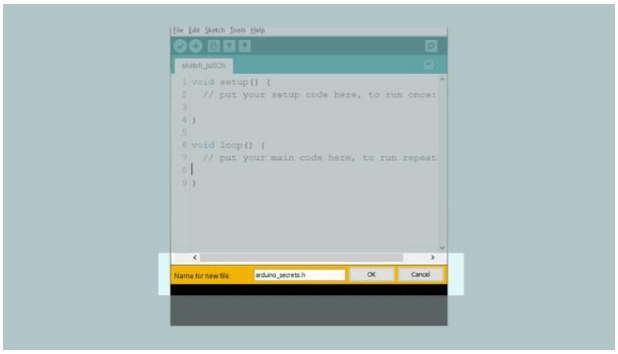
   arduino_secrets.h

   that we can store our Wi-Fi credentials in. To create a tab in the offline editor, click the arrow symbol underneath the Serial Monitor symbol, and click on the "New tab" option.



Creating a new tab.

Then, name the file "arduino_secrets.h".



Naming the file.

1. Inside this new header file, we need to use the below code, where our

   SECRET_SSID

   (network name) and

   SECRET_PASS

(password) needs to be replaced by your own credentials.

COPY

```
1#define SECRET_SSID ""

2#define SECRET_PASS ""
```

6. We can now copy and paste the **sender** code below into our regular sketch file, and upload it to our board. Make sure we have selected the right port and board before uploading.

Note: The char

topic[]
,

topic2[]
and

topic3[]
, created here may be used by someone else. If we change this, we will also need to change the name of the topic we subscribe to in the *subscriber sketch*.

COPY

```
1#include <ArduinoMqttClient.h>

2#include <WiFiNINA.h>

3#include "arduino_secrets.h"

4

5///////please enter your sensitive data in the Secret tab/arduino_secrets.h

6char ssid[] = SECRET_SSID;        // your network SSID (name)

7char pass[] = SECRET_PASS;    // your network password (use for WPA, or use as key for WEP)

8

9WiFiClient wifiClient;

10MqttClient mqttClient(wifiClient);

11

12const char broker[] = "test.mosquitto.org";

13int        port     = 1883;
```

```
14const char topic[] = "real_unique_topic";
15const char topic2[] = "real_unique_topic_2";
16const char topic3[] = "real_unique_topic_3";
17
18//set interval for sending messages (milliseconds)
19const long interval = 8000;
20unsigned long previousMillis = 0;
21
22int count = 0;
23
24void setup() {
25  //Initialize serial and wait for port to open:
26  Serial.begin(9600);
27  while (!Serial) {
28    ; // wait for serial port to connect. Needed for native USB port only
29 }
30
31  // attempt to connect to Wifi network:
32  Serial.print("Attempting to connect to WPA SSID: ");
33  Serial.println(ssid);
34  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
35    // failed, retry
36    Serial.print(".");
37    delay(5000);
38 }
39
40  Serial.println("You're connected to the network");
```

```
41  Serial.println();
42
43  Serial.print("Attempting to connect to the MQTT broker: ");
44  Serial.println(broker);
45
46  if (!mqttClient.connect(broker, port)) {
47    Serial.print("MQTT connection failed! Error code = ");
48    Serial.println(mqttClient.connectError());
49
50    while (1);
51 }
52
53  Serial.println("You're connected to the MQTT broker!");
54  Serial.println();
55}
56
57void loop() {
58  // call poll() regularly to allow the library to send MQTT keep alive which
59  // avoids being disconnected by the broker
60  mqttClient.poll();
61
62 unsigned long currentMillis = millis();
63
64  if (currentMillis - previousMillis >= interval) {
65    // save the last time a message was sent
66    previousMillis = currentMillis;
67
```

```
68  //record random value from A0, A1 and A2
69  int Rvalue = analogRead(A0);
70  int Rvalue2 = analogRead(A1);
71  int Rvalue3 = analogRead(A2);
72
73  Serial.print("Sending message to topic: ");
74  Serial.println(topic);
75  Serial.println(Rvalue);
76
77  Serial.print("Sending message to topic: ");
78  Serial.println(topic2);
79  Serial.println(Rvalue2);
80
81  Serial.print("Sending message to topic: ");
82  Serial.println(topic2);
83  Serial.println(Rvalue3);
84
85  // send message, the Print interface can be used to set the message contents
86  mqttClient.beginMessage(topic);
87  mqttClient.print(Rvalue);
88  mqttClient.endMessage();
89
90  mqttClient.beginMessage(topic2);
91  mqttClient.print(Rvalue2);
92  mqttClient.endMessage();
93
94  mqttClient.beginMessage(topic3);
```

```
95    mqttClient.print(Rvalue3);
96    mqttClient.endMessage();
97
98    Serial.println();
99 }
100}
```

Programming the Subscriber Device

We will now program the subscriber device. For this, we need to create a new sketch, and create another

arduino_secrets.h
file.

8. We can now copy and paste the **receiver** code below into our regular sketch file, and upload it to our board. Make sure we have selected the right port and board before uploading.

COPY

```
1#include <ArduinoMqttClient.h>
2#include <WiFiNINA.h>
3#include "arduino_secrets.h"
4
5///////please enter your sensitive data in the Secret tab/arduino_secrets.h
6char ssid[] = SECRET_SSID;        // your network SSID
7char pass[] = SECRET_PASS;    // your network password
8
9WiFiClient wifiClient;
10MqttClient mqttClient(wifiClient);
11
12const char broker[] = "test.mosquitto.org";
13int        port     = 1883;
```

```
14const char topic[] = "real_unique_topic";
15const char topic2[] = "real_unique_topic_2";
16const char topic3[] = "real_unique_topic_3";
17
18void setup() {
19  //Initialize serial and wait for port to open:
20  Serial.begin(9600);
21  while (!Serial) {
22    ; // wait for serial port to connect. Needed for native USB port only
23 }
24  // attempt to connect to Wifi network:
25  Serial.print("Attempting to connect to SSID: ");
26  Serial.println(ssid);
27  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
28    // failed, retry
29    Serial.print(".");
30    delay(5000);
31 }
32
33  Serial.println("You're connected to the network");
34  Serial.println();
35
36  Serial.print("Attempting to connect to the MQTT broker: ");
37  Serial.println(broker);
38
39  if (!mqttClient.connect(broker, port)) {
40    Serial.print("MQTT connection failed! Error code = ");
```

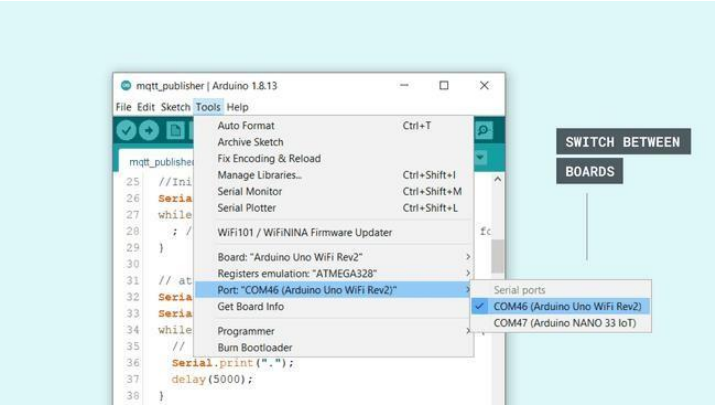
```
41    Serial.println(mqttClient.connectError());
42
43    while (1);
44 }
45
46  Serial.println("You're connected to the MQTT broker!");
47  Serial.println();
48
49  // set the message receive callback
50  mqttClient.onMessage(onMqttMessage);
51
52  Serial.print("Subscribing to topic: ");
53  Serial.println(topic);
54  Serial.println();
55
56  // subscribe to a topic
57  mqttClient.subscribe(topic);
58  mqttClient.subscribe(topic2);
59  mqttClient.subscribe(topic3);
60
61  // topics can be unsubscribed using:
62  // mqttClient.unsubscribe(topic);
63
64  Serial.print("Topic: ");
65  Serial.println(topic);
66  Serial.print("Topic: ");
67  Serial.println(topic2);
```

```
68  Serial.print("Topic: ");
69  Serial.println(topic3);
70
71 Serial.println();
72}
73
74void loop() {
75  // call poll() regularly to allow the library to receive MQTT messages and
76  // send MQTT keep alive which avoids being disconnected by the broker
77  mqttClient.poll();
78}
79
80void onMqttMessage(int messageSize) {
81  // we received a message, print out the topic and contents
82  Serial.println("Received a message with topic '");
83  Serial.print(mqttClient.messageTopic());
84  Serial.print("', length ");
85  Serial.print(messageSize);
86  Serial.println(" bytes:");
87
88  // use the Stream interface to print the contents
89  while (mqttClient.available()) {
90    Serial.print((char)mqttClient.read());
91 }
92  Serial.println();
93  Serial.println();
94}
```

Testing It Out

If everything was successful during the upload, we now have a **publisher** and **subscriber** device. Next, we need to open the Serial Monitor for each board, one at a time. This will initialize the sketch. Since we can only have one Serial Monitor open at one time, we will need to switch the ports manually. Using only one computer can be a bit tedious, as we can never view the Serial Monitor of both devices at the same time.

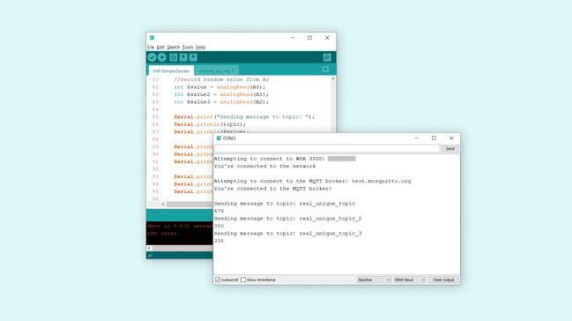
In this tutorial, a Arduino UNO WiFi Rev2 and a Nano 33 IoT board was used. When switching between the ports, we can see them listed as COM12 and COM3.



Switching between boards.

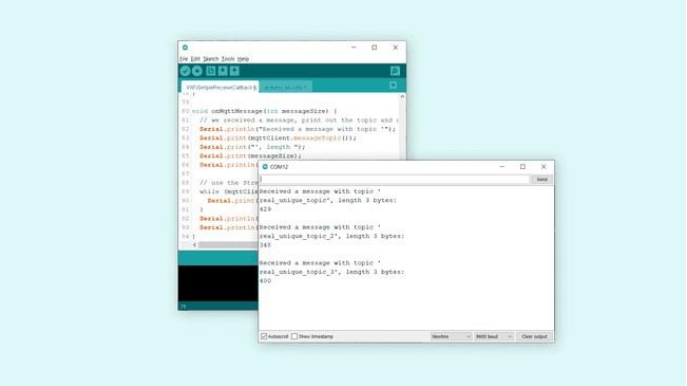
When both devices have been initialized, they start connecting to the network. Now, let's open the Serial Monitor and keep it open for the **publisher** device.

We can now see that we are sending messages every 8 seconds (this interval can be changed at the top of the code). Each interval sends three messages, for each topic. Each topic contains the latest reading from an analog pin.



Serial Monitor of the publisher device.

Now, let's open the **subscriber** device's Serial Monitor, and keep it open. If everything is successful, we can now see the analog values. This means, that we have successfully published three topics on one device, and subscribed to them using another device.



Serial Monitor of the subscriber device.

Troubleshoot

If the code is not working, there are some common issues we can troubleshoot:

- Check that the host is right: this tutorial uses **test.mosquitto.org**.
- Check that our credentials are correct in the

  arduino_secrets.h

  tab.

- Make sure that the topics we publish match the topics we subscribe to.

Conclusion

In this tutorial, we have created a very basic MQTT application, which allows data to flow from a publisher device, via a broker, to a subscriber device. This type of setup is commonly used in many Internet of Things (IoT) applications, and we encourage you to continue exploring the **ArduinoMqttClient** library. In this tutorial, in order to create a minimal working project we did not use any form of encryption. This is OK to use for just basic, non-private data, but if you are intending to use it for e.g. home automation, security systems and so on, it is **strongly recommended** to use more security layers.

The broker used in this tutorial, **test.mosquitto.org** provides some explanation on different forms of encryption, and what ports to use etc.