Introduction to Arduino IDE

- Presentation
- Getting started
- Some basics
- First steps
  - Import libraries
  - Define constants
  - Define global variables
  - Setup function
  - Display function
  - Sending function
  - Receiving function
  - Loop function
  - Uploading a program
- Conclusion

Presentation

Arduino IDE is a software used to write code and upload programs to Arduino compatible boards. It is able to run on Windows, Mac OS X and Linux. It also proposes an online version which allows you to develop your sketch and then store it in the cloud. The Arduino language is based on the C and C++ language.

Getting started

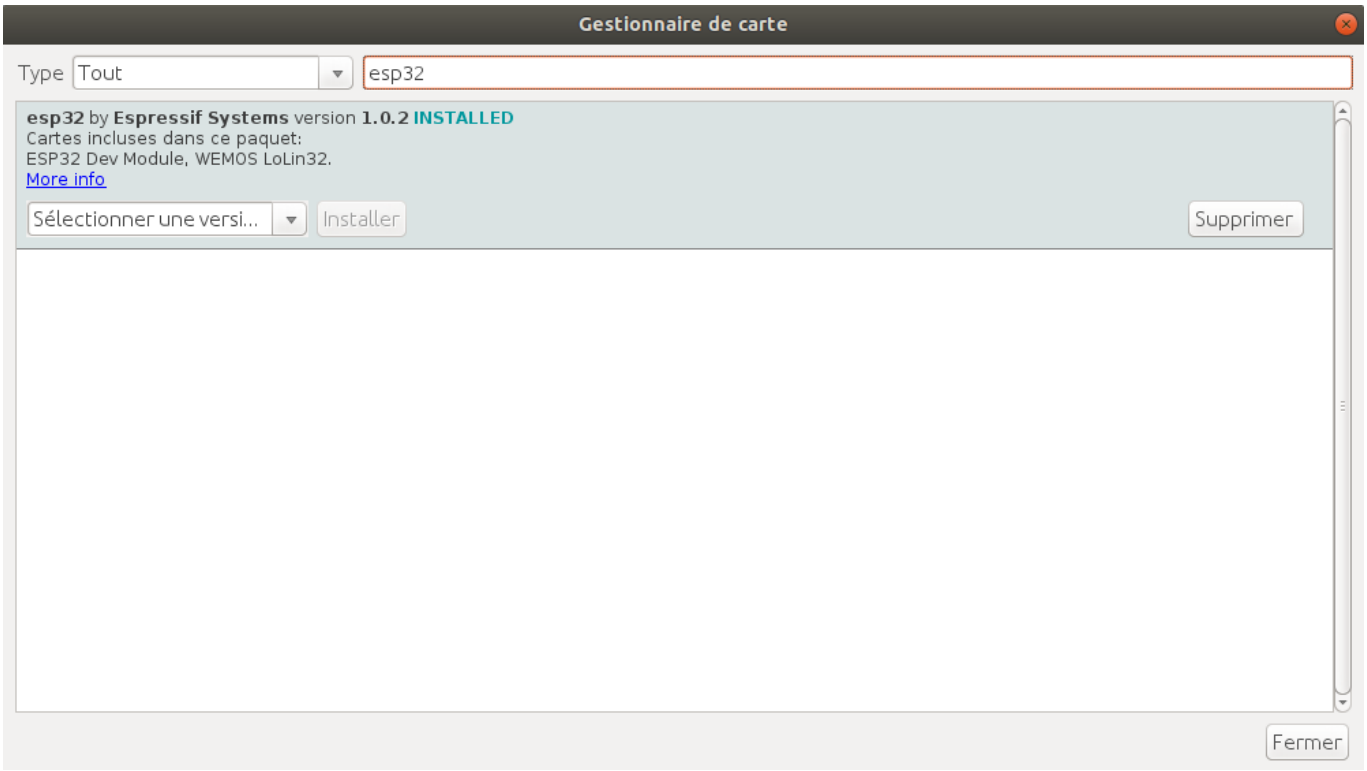
First of all, you will have to download the Arduino IDE if it is not already installed on the computer. To do so, you will have to go to their official website : https://www.arduino.cc/en/Main/Software

Download Page

Notes : if you have some troubles, you can refer to the Getting Started page

(https://www.arduino.cc/en/Guide/HomePage).

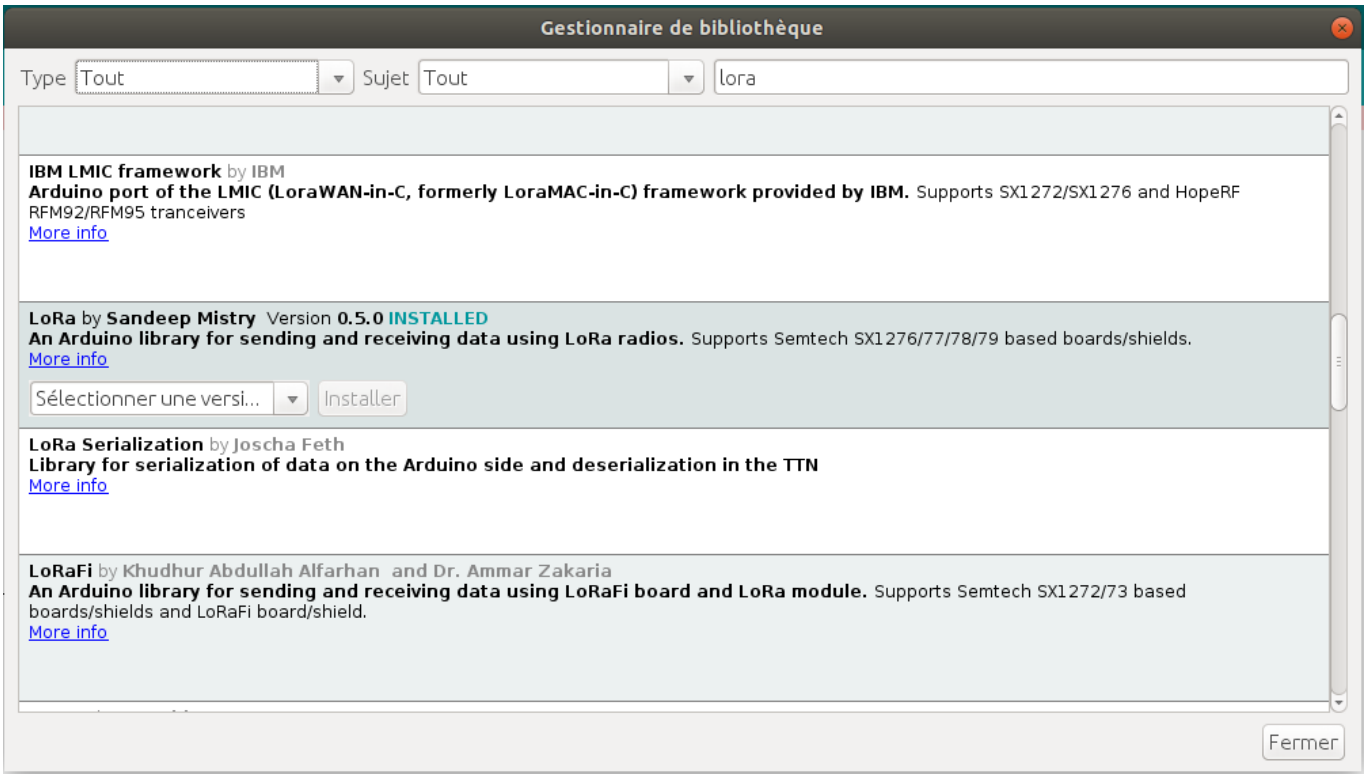
Once the IDE is installed, you need to install a package that is required for esp32 boards. This package that you need to install is esp32 so that the card is recognized by the IDE. To do so, you will need to go to **File > Preferences** and add this URL : https://dl.espressif.com/dl/package_esp32_index.json into *Additional Board Manager URLs* field. Once it is done, go to **Tools > Board > Boards Manager**. Then search for **esp32** and install it. After that you should be able to select the type of the card : **Tools > Type of card > ESP32 Dev Module**.



ESP32

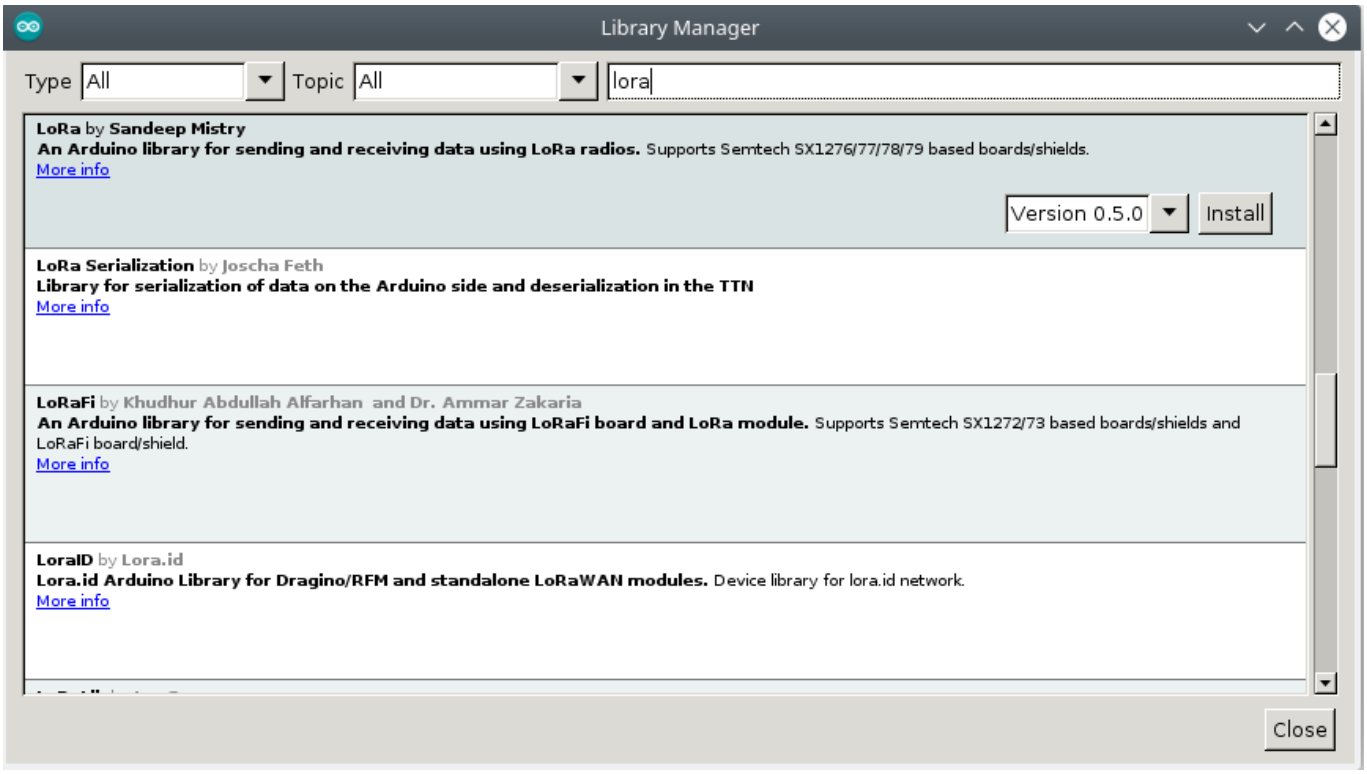
Note : If you get an issue to compile due to `No module named serial`. It means that pyserial is probably missing. To solve this problem, you will have to install it with : `pip install pyserial`.

You will have to install a library required for the use of LoRa. To do so, go to **Sketch > Include library > Manage Libraries**.



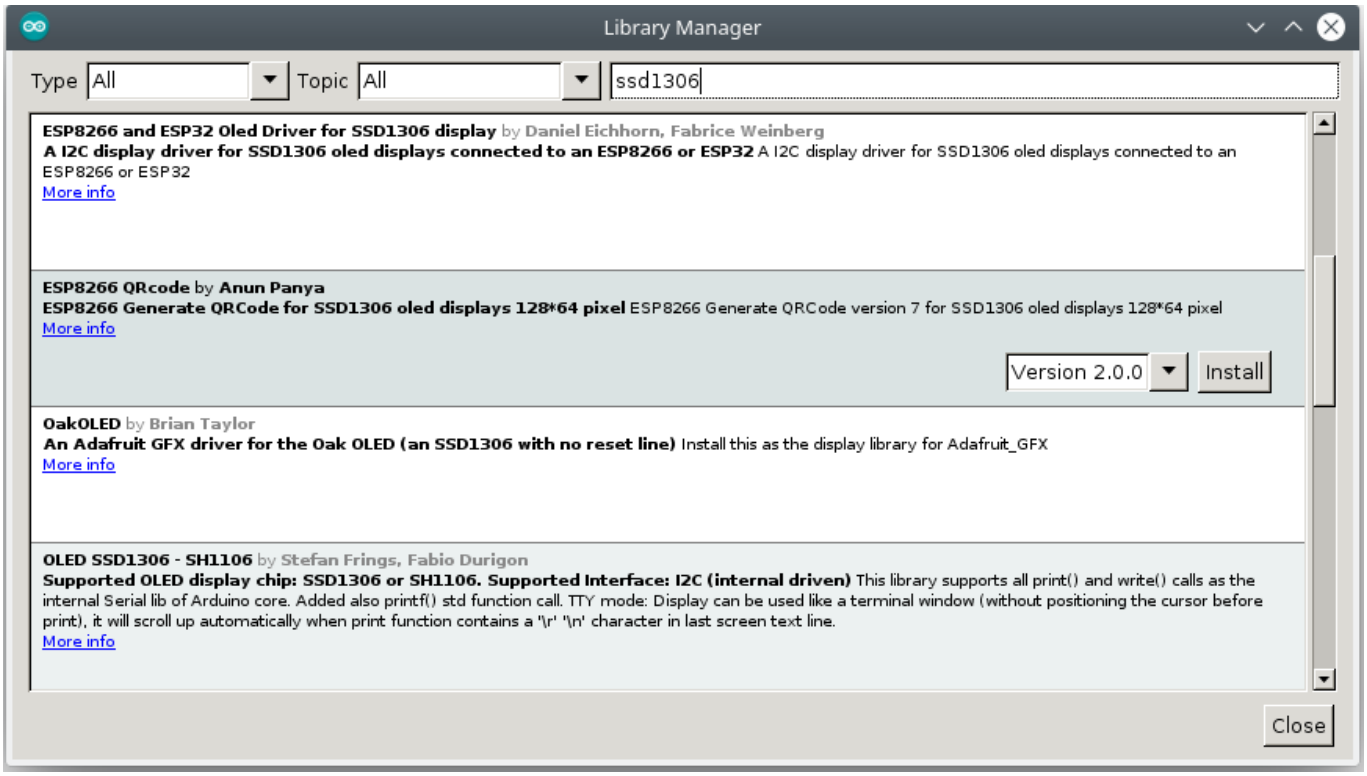
Manage libraries

Then search for **LoRa** library by *Sandeep Mestry*.



LoRa library

Last step is to install the library to be able to use the OLED screen. You have to search in the libraries and install **ESP8266 and ESP32 Oled Driver for SSD1306 display** *by Daniel Eichhorn and Fabrice Weinberg*.



Oled library

After that you can start to code.

Some basics

When you launch the IDE, the empty sketch contains already two default functions : the setup function and the loop function.

Default sketch

- Setup : This function allows to initialize the program and to set initial values. It runs only once at the beginning of the program.

- Loop : This function defines the actions that will be executed as long as the card is powered.

You can also write other functions in order to organize your code.

First steps

You will have to create two different files. One will be uploaded to the sender device and the other will be uploaded to the receiver device. The import of libraries, the variable declaration, the setup function and the display function will be the same for both.

Import libraries

Now that you have installed the libraries, you need to import them in order to use them. To do so :

```
#include <SPI.h>
#include <LoRa.h>
#include <SSD1306.h>
```

- **SPI** : It is a default library that allow some device to communicate. More information here https://www.arduino.cc/en/Reference/SPI.
- **LoRa** : It is the library used for sending and receiving packet via LoRa radio. More information here https://github.com/sandeepmistry/arduino-LoRa.
- **SSD1306** : It is the library that have some built-in functions to configure Oled screen. More information here https://github.com/ThingPulse/esp8266-oled-ssd1306.

Define constants

Now that you have imported all the libraries you will need, you can define some constants related to GPIO. GPIO stands for General Purpose of Input Output. We just give an alias to pins that will be used in certain functions in the program.

```
#define SCK     5 //Serial Clock to synchronize data transmition
#define MISO    19 //The slave line for sending data to the master
#define MOSI    27 //The master line for sending data to the peripherals
#define SS      18 //The pin on each device that the master can use to
enable or disable specific devices
#define RST     14 //Reset pin
#define DI0     26
#define freq    433E6 //Frequency used by LoRa network 433
#define sf 12 //Must be between 6 and 12. The larger the number, the
farther the package can go.
#define sb 125E3 //Signal bandwidth
#define NUMBER_OF_DISPLAY_LINES 2
#define NUMBER_OF_CHARS_PER_LINE 22
#define NUMBER_OF_CHARS_WITH_ZERO (NUMBER_OF_CHARS_PER_LINE+1)
```

In this case you are giving the value 5 to the constant SCK. The difference between a variable and a constant is that the constant will not take up any memory space. It will be replaced in the source code during precompilation phase.

Define global variables

You need to define some variables then :

```
SSD1306  display(0x3c, 4, 15); //Declaration of the variable to configure
the screen
char displayBuffer[NUMBER_OF_DISPLAY_LINES*NUMBER_OF_CHARS_WITH_ZERO];
//This variable will keep in memory all the information to be displayed
int cpt = 1; //This variable will count the number of packets sent
char line[30]; //This variable will store the message that you will send

union pack
{
uint8_t frame[16];
char data[50];
} sdp ; //You define a new type that combines other types and declare a
variable 'sdp'
```

Setup function

You will be able to initialize the program as you have already defined all variables. To do this, you will define the setup function as follows :

- Display configuration
- Hardware configuration
- LoRa configuration

So let's start with display configuration.

```
memset(displayBuffer,0,NUMBER_OF_DISPLAY_LINES*NUMBER_OF_CHARS_WITH_ZERO);
//Fill the displayBuffer with 0
display.init(); //Initialize the Oled screen
display.flipScreenVertically(); //Turn the display to landscape mode
display.setFont(ArialMT_Plain_10); //Set a specific font
display.setTextAlignment(TEXT_ALIGN_LEFT); //Set the text alignment
```

Once the screen is configured, you have to configure the hardware.

```
pinMode(16,OUTPUT); //Define the pin 16 to behave as an output
digitalWrite(16, LOW);    //Set GPIO16 low to reset OLED
delay(50);
digitalWrite(16, HIGH); //Set GPIO16 high to switch on OLED
Serial.begin(9600); //Set the datarate in bit/s
```

```
pinMode(DI0,OUTPUT); //Define the pin 26 to behave as an output
SPI.begin(SCK,MISO,MOSI,SS);
```

Finally, you have to setup the LoRa characteristics.

```
LoRa.setPins(SS,RST,DI0); //Match some pins
if (!LoRa.begin(freq)) { //Try to initialise LoRa with a specific frequency
    Serial.println("Starting LoRa failed!");
    while (1);
}
LoRa.setSpreadingFactor(sf); //Change the spreading factor
LoRa.setSignalBandwidth(sb); //Setup the bandwidth
```

Note : You can use `LoRa.setSyncWord(0xFF);` to specify a sync word in order not to receive messages from foreign senders. The range goes from 0 to 0xFF. Replace 0xFF by your own sync word.

Display function

This function will write on the Oled screen all the information you want.

```
void outputDisplayLine(unsigned char lineNumber, char *string) {
  strncpy(displayBuffer+lineNumber*NUMBER_OF_CHARS_WITH_ZERO, string,
NUMBER_OF_CHARS_PER_LINE);

  display.clear();
  for(int i=0; i<NUMBER_OF_DISPLAY_LINES; i++) {
    display.drawString(1,13*i,displayBuffer+i*NUMBER_OF_CHARS_WITH_ZERO);
  }
  display.display();
}
```

Sending function

You will use functions that are defined in the LoRa library in order to send a packet.

```
void sendPacket(int n){
  LoRa.beginPacket(); //Initialize a new packet
  sprintf(sdp.data, "packet n°%d", cpt); //Add in the data part our message
  LoRa.write(sdp.frame,50);
  LoRa.endPacket();
  sprintf(line,"Packet n°%d sent! \n",cpt); //Write some logs in the Serial
monitor
  outputDisplayLine(1,line);
  Serial.write(line);
}
```

Receiving function

By default, the reicever will be in a listening status. Once it receives a package, this function gets data and displays it on the screen.

```
void receivePacket(){
    while (LoRa.available()) {
      String data = LoRa.readString(); //Read the data
      line = strdup(data.c_str());
      outputDisplayLine(1,line); //Display data on screen
    }
}
```

Loop function

There will be two loop functions, one for the sender and one for the receiver

Sender code :

```
void loop(){
  outputDisplayLine(0,"LoRa Ping Sender"); //Display on the screen Sender
  sendPacket(cpt); //Send a packet with the counter to specify packet
number
  cpt++; //Increment the counter
  delay(10000);
}
```

Receiver code :

```
void loop(){
  outputDisplayLine(0,"LoRa Ping Receiver"); //Display on the screen
receiver
  packetSize = LoRa.parsePacket(); //Get the packetsize
  if (packetSize){ //If packetSize has a valid value, it means that a
packet has been received
    receivePacket();
    delay(7000);
  }
  else{
    outputDisplayLine(1,"Nothing received"); //Else, display that no packet
has been received
  }
}
```

Uploading a program

Now that you have finished coding, you need to check if the code does not contain any error and if so, you can deploy it on the card. To do so, there are two buttons under **File** menu.

Left : Validate button | Right : Upload button

Conclusion

This document provides a minimalist example of a LoRa sender / receiver. With the library, you can specify more parameters such as : txpower, coding rate etc... To go further, you can try to implement an echo reply scenario in which the receiver sends a message to confirm that it has received the message.