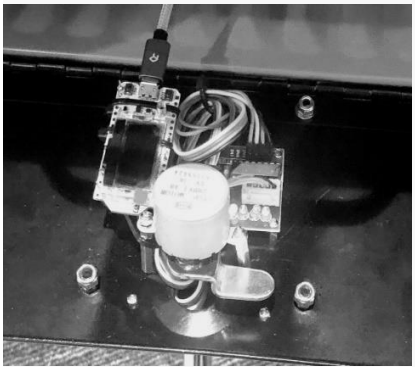
Smart Lock Box Starter Kit



Hardware you will Need for this kit

Heltec WiFi LoRa 32 (V2)28BYJ-48

Adafruit Industries 12VDC 32-Step Small Reduction Stepper Motor

LoRa GPS HAT for Raspberry Pi

4x4 Matrix keypad

Raspberry Pi 3 or 4

Arduino Libraries needed:

- LoRa
- QRCode
- Keypad
- AccelStepper
- U8g2

Raspberry Pi Libraries needed (pip) :

For the Starter Kit SDK (setup.sh will install these):

- mnemonic
- pycrypto
- pure25519
- paho-mqtt
- redis
- requests
- stellar-sdk
- toml

For the Gateway:

- pyqrcode
- Tkinter
- ed25519

Pins for motor:

- IN1 = 12
- IN2 = 13
- IN3 = 2
- IN4 = 17

Pins for keypad:

- Rows - 23, 32
- Columns - 22, 33

Introduction on the 28BYJ-48 stepper motor and the ULN2003 driver board

The **28BYJ-48 stepper motor** is widely used to control a myriad of common devices we see every day. From blinds, car side mirror tilts and DVD players to security cameras and precise control machines, stepper motors are quite popular, we just don't realize it.

We have tried our hand at simplifying that information in this summary for you, on how to interface a 28BYJ-48 stepper motor with an Arduino, using a ULN2003 driver board.

What is the 28BYJ-48 Stepper motor?

The 28BYJ-48 stepper motor is a commonly used stepper motor, which converts electrical pulses into discrete mechanical rotation.

Why is the name stepper used?

When electrical signals are applied, the stepper motor rotates in accurate and fixed angle increments known as steps. The motor consists of 4 coils that make a ring around the rotor. These coils are known as the stator, as they are stationary and static. Each coil is rated at +5V, making it easy to control with any microcontroller, such as an Arduino.

What is the difference between stepper motors and standard DC motors?

Why do we need the 28BYJ-48 stepper motor?
Before deciding on which type of motor is more suitable for your project, we will have to first understand some differences between the two motors.

Here are some basic differences between the two.

1. The rotation of stepper motors is incremental, slow and precise, while DC motors have a fast, continuous motion.

2. Stepper motors are known to generate some noise during operation while DC motors are quiet and relatively vibration free.

3. The response time of the stepper motor is slower than the DC motor.

4. Stepper motors can be easily controlled with microprocessors like the Arduino. Compared to DC motors, they are more mechanically simple and easy to design and build. In contrast, DC motors are not so easily controlled with microprocessors

What is the difference between stepper motors and servo motors?

You could use a servo motor, but should you ?

1. Stepper motors can move more accurately and precisely than the servo motor, and are much more easier to control.

2. Stepper motors are more suitable for applications with lower speeds of less than 2000 rpm (revolutions per minute), and servos are best suited for applications with high speeds greater than 2000 rpm.

3. Stepper motors operate at lower speeds than servo motors.

4. Stepper motors are cheaper than servo motors as they are less mechanically complex.

When would I choose a stepper motor over the other motor types?

Here are some areas where the stepper motor triumphs over the other two :

Precise Positioning

- Stepper motors move in precise steps.
  - they do well in applications that require precise positioning
    - like 3D printers and camera platforms

Precise Speed control

- Precise increments in movements enable excellent control of rotational speed for process automation and robotics

High Torque at low speeds

- Stepper motors are best suited for applications with low speed
  - Less than 2000 rpm
  - they have maximum torque at low speeds
- In contrast, normal DC motors and servo motors do not have so much torque at low speeds

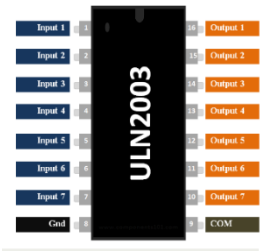
Easy to control

- Stepper motors can easily be controlled using a microcontroller like an Arduino
- Ease of use a major reason for its continued usage by stepper motor users

Cheap

- In applications where stepper motors would suffice, and using one could help you to realize better cost savings

What is the ULN2003 Driver Board?

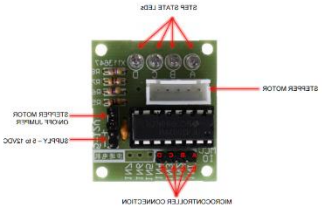


The ULN2003 is one of the most common motor driver ICs that houses an array of 7 Darlington transistor pairs, each capable of driving loads up to 500mA and 50V.

Basically, a Darlington pair is a pair of transistors, where the second transistor amplifies the output current of the first transistor.

The ULN2003 IC is needed to drive the motor with an Arduino, but they come in convenient and cheap driver boards that are readily available complete with indicator LEDs.

So it's better to grab one of these rather than breadboarding the IC itself.



As shown in the diagrams above, ULN2003 driver board consists of

- ULN2003 soldered onto a board
- together with resistors, capacitors, and other bits and bobs
- that help create the circuit that takes the pulse signals from the controller and converts them into stepper motor motion

Why is the ULN2003 needed?

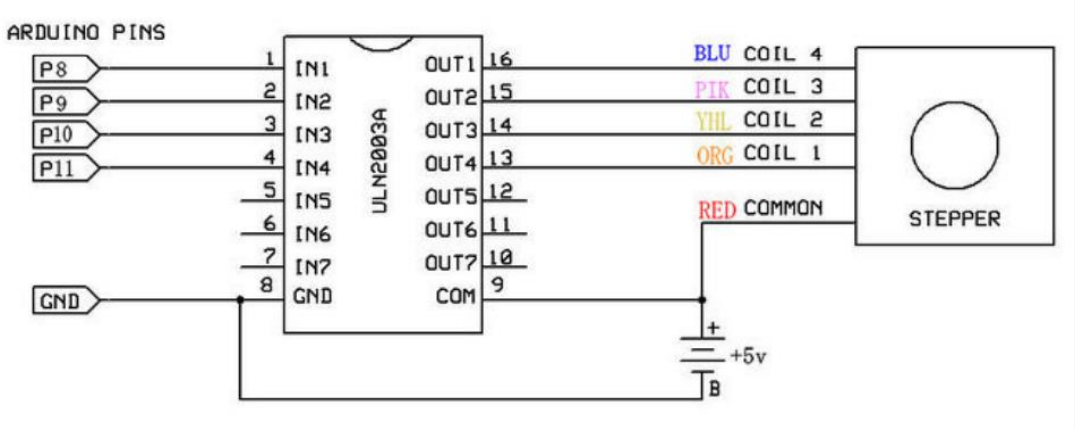
The 28BYJ-48 stepper motor consumes high current and hence, we will need to use a driver IC like the ULN2003 in order to control the motor with a microcontroller like the Arduino.

Known for its high current and high voltage capacity, the ULN2003 gives a higher current gain than a single transistor and enables the low voltage and low current output of a microcontroller to drive a higher current stepper motor.

For example, a stepper motor that needs 9V and 300mA to operate cannot be powered by an Arduino. So we connect this IC to source for enough current and voltage for the motor.

If you have to power anything more than 5V and 80mA, the ULN2003 driver board should be used.

Take note that powering the stepper motor directly from the Arduino is not recommended.



Connecting the 28BYJ-48 stepper motor to the ULN2003 driver board

Usually, the 28BYJ-48 stepper motor comes with a 5-pin connector that will fit to the ULN2003 driver board

Connecting the ULN2003 driver board to the Arduino

Connect the ULN2003 driver IN1, IN2, IN3, IN4 to the Arduino digital pins 8, 9, 10, and 11 respectively.

The driver board has two pins which are labeled GND and VCC, which are two pins for the power supply.

The ULN2003 driver board's GND pin must be connected to the Arduino's GND pin. Similarly, the driver board's VCC pin should be connected to the Arduino's 5V pin.

Important note:

Using this setup, we are powering the stepper motor directly from the Arduino. Although this is the simplest way to provide power to the motor, it is not recommended to do so (connecting the driver board VCC pin to the 5V pin of the Arduino).

This is because if the motor draws too much power, your Arduino can potentially be damaged.

Make sure not more than 300mA is drawn out of your Arduino. If more power is needed, simply connect your driver board to an external voltage supply rather than using the Arduino's onboard power supply.

We will be powering the motor from the Arduino in this kit to keep the connections simple.

Once you have familiarized yourself with the connections, be sure to connect your motor to an external power source instead if more power is needed.

Example Code

The Arduino Integrated Development Environment (IDE), or Arduino software, comes pre-installed with a default stepper library. Hence, there is no need to download the library. On the first line of the code, we will first include the header file of this built-in stepper library.

We define the number of steps that will be made for one revolution.

Working out this number might be a little tricky, so here's how it's done.

Stepper motors can be driven in different modes and they have a specific gear ratio.

Both factors have an influence on the number of steps per revolution.

For this example, we shall drive the motor in a mode known as the full step mode, with each step corresponding to a rotation of 11.25 degrees according to the datasheet.

That means there are 32 steps per revolution (360/11.25 = 32).

In addition, the manufacturer has specified a gear ratio of 64:1 for the 28BYJ-48 stepper motor.

To obtain the final number of steps, the gear ratio must be multiplied by the number of steps per revolution, 32.

The more accurate value for the gear ratio is actually about 63.68395 : 1.

Therefore, we set the final number of steps to 2038 (32 x 63.68395 = 2037.8864).

Next, we initialize the stepper.

Within the brackets are the parameters.

The first parameter is the number of steps.

As it was already defined in the previous line, we can write STEPS to represent this number.

The other parameters correspond to the Arduino pins that are used to connect the ULN2003 driver board.

As stated previously, the pins are 8, 9, 10 and 11.

The next loop function is where we will write our code to drive the motor.

For the first line of the loop, we set the speed of one revolution to one per minute.
Next, we tell the stepper motor to do 2038 steps.

Since one revolution corresponds to 2038 steps as we have calculated previously, that means the motor shaft should move a full revolution within about one minute.

Next, we set a delay of one second.

In the next two lines, we do the same thing again – setting the number of revolutions per minute and commanding the stepper to do a number of steps.

However, this time, we set the speed to 6 rounds per minute and move the shaft in the other direction by setting a negative number of steps.

This means that the motor will move 6 times faster, and should complete a full revolution in about 10s (60s / 6 = 10s).

Here's the example code.

Upload the following code into your Arduino programme and try it out

```
#include <Stepper.h>
#define STEPS 2038 // the number of steps in one revolution of your motor (28BYJ-48)
Stepper stepper(STEPS, 8, 10, 9, 11);
void setup() {
// nothing to do
}
void loop() {
stepper.setSpeed(1); // 1 rpm
stepper.step(2038); // do 2038 steps -- corresponds to one revolution in one minute
delay(1000); // wait for one second
stepper.setSpeed(6); // 6 rpm
stepper.step(-2038); // do 2038 steps in the other direction with faster speed
-- corresponds to one revolution in 10 seconds
}
```

Arduino Keypad:

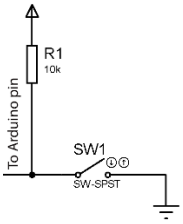
Interfacing with 4x4 Matrix

Using a keypad is an upgrade over using buttons for input on your Arduino project.

A keypad is a set of 12 or 16 buttons wired so that the pin usage is reduced.

Button vs Keypad :

Wiring one button to the Arduino requires using one of its pins:

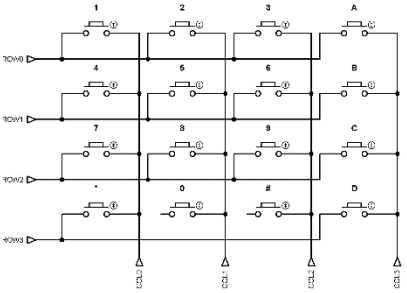


So if you would need 16 different buttons,

you may need to use 16 Arduino pins.

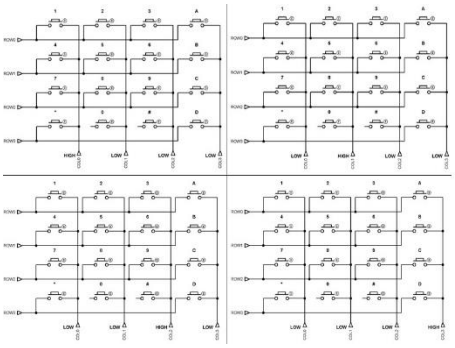
A keypad matrix is wired differently:

A 4x4 keypad matrix has 8 pins divided into 4 rows and 4 columns.



When a button is pressed, one row pin will be shorted out with a column pin.

For example, if you press button “1”, row “0” will be connected to column “0”.



How a Keypad Matrix Works

Now to be able to make the keypad matrix work with an Arduino, we just need to check which row and column is connected.

The most common way to implement this by making a column pin high while making the rest of the columns low and do that in sequence like a LED chaser.

Consider the first image (upper left) on the panel above where column “0” is high and the rest of the columns are low.

If you press the “1” button, row “0” will be high because by then it will be connected to column “0”.

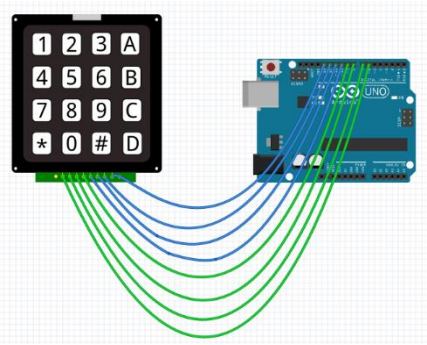
So if you scan row “0” using digitalWrite() while column “0” is high, this means the user pressed button “1”.

To detect every button that is pressed, we also need to scan each rows individually.

So basically, you can summarize the process like this:

When this button is pressed...	If...	Then...
1	Col0=high	Row0=high
2	Col1=high	Row0=high
3	Col2=high	Row0=high
4	Col0=high	Row1=high
5	Col1=high	Row1=high
6	Col2=high	Row1=high
7	Col0=high	Row2=high
8	Col1=high	Row2=high
9	Col2=high	Row2=high
0	Col1=high	Row3=high
*	Col0=high	Row3=high
#	Col2=high	Row3=high
A	Col3=high	Row0=high
B	Col3=high	Row1=high
C	Col3=high	Row2=high
D	Col3=high	Row3=high

Keypad Scanning with Arduino



The rows are connected to digital pins 10 to 13 while the columns pins are connected to digital pins 6 to 9.

Although it's official, the Keypad library is not installed on the IDE by default.

To install the library, go to Sketch > Include Library > Manage Libraries.

Type "keypad" on the search box and locate the library by Mark Stanley and Alexander Brevig.

Once installed, the Keypad folder should now appear on the examples menu.

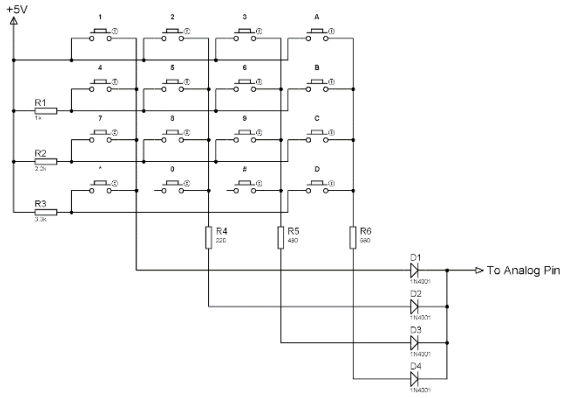
There are a lot of examples provided in this folder. Try the CustomKeyboard sketch.

Using Fewer Pins with Keypad Matrix

There is another way to use the 4x4 keypad matrix without using 8 Arduino pins.

In fact, you'll only be using one pin.

All you need is wire the matrix like this:



Using this schematic,

when a button is pressed,

a different voltage level is read on the output pin.

For example, if button "4" is pressed, current will flow from the +5V source, through the 1k resistor and through the diode.

The voltage at the output will $V_{out} = 5 - V_{res} - 0.7$ be

This is the rendered form of the equation.

If button "8" is pressed, the current will flow through the 2.2K and 220 resistor, giving a different voltage at the output and so on.

The downside of using this method is you have to test each but

ton and acquire their output voltage as computations will be different with the actual value.

Use the built-in AnalogReadSerial example (Files > Examples > 01.Basics > AnalogReadSerial) for testing while connecting the output pin in the schematic above to A0.

4x3 keypad matrix instead of the 4x4 matrix

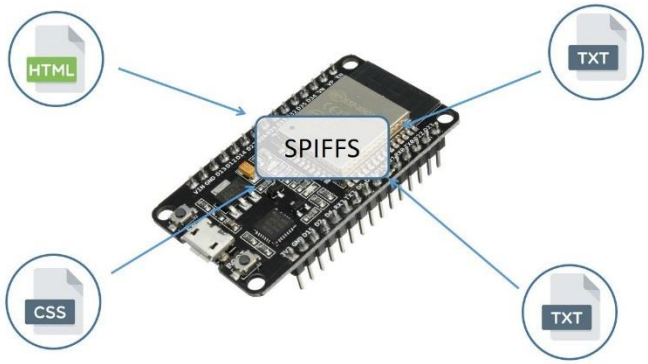
Just remove the fourth column

Introducing SPIFFS

SPIFFS lets you access the flash memory like you would do in a normal filesystem in your computer, but simpler and more limited.

You can read, write, close, and delete files.

At the time of writing this post, SPIFFS doesn't support directories, so everything is saved on a flat structure.



Installing ESP32 Filesystem Uploader in Arduino IDE

The ESP32 contains a Serial Peripheral Interface Flash File System (SPIFFS).

SPIFFS is a lightweight filesystem created for microcontrollers with a flash chip, which are connected by SPI bus, like the ESP32 flash memory.

Using SPIFFS with the ESP32 board is specially useful to:

- Create configuration files with settings;
- Save data permanently;
- Create files to save small amounts of data instead of using a microSD card;
- Save HTML and CSS files to build a web server;
- Save images, figures and icons;
- And much more.

In many projects, you'd have written the HTML code for the web server as a String directly on the Arduino sketch.

With SPIFFS, you can write the HTML and CSS in a separated file and save them on the ESP32 filesystem.

Installing the Arduino ESP32 Filesystem Uploader

You can create, save and write files to the ESP32 filesystem by writing the code yourself on the Arduino IDE.

This is not very useful, because you'd have to type the content of your files in the Arduino sketch.

But there is a plugin for the Arduino IDE that allows you to upload files directly to the ESP32 filesystem from a folder in your computer.

This makes it really easy and simple to work with files. Let's install it.

First, make sure you have the latest Arduino IDE installed, and you have the ESP32 add-on for the Arduino IDE.

If you don't, follow one of the following tutorials to install the add-on:

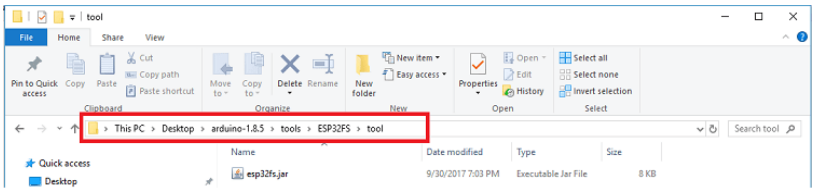
https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

Follow the next steps to install the filesystem uploader:

1) Go to the releases page and click the ESP32FS-1.0.zip file to download.

2) Go to the Arduino IDE directory, and open the Tools folder.

3) Unzip the downloaded .zip folder to the Tools folder.



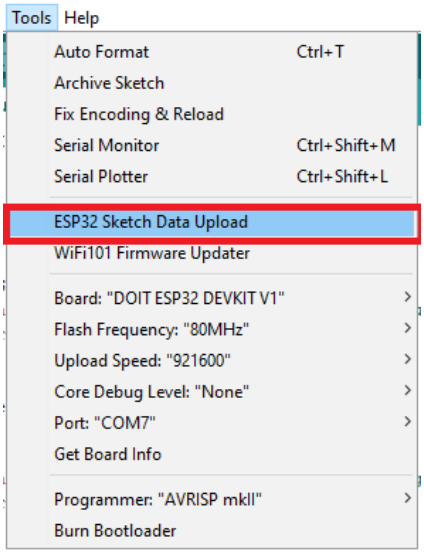
<home_dir>/Arduino-<version>/tools/ESP32FS/tool/esp32fs.jar

4) Finally, restart your Arduino IDE.

To check if the plugin was successfully installed, open your Arduino IDE.

Select your ESP32 board,

go to Tools and check that you have the option “ESP32 Sketch Data Upload“



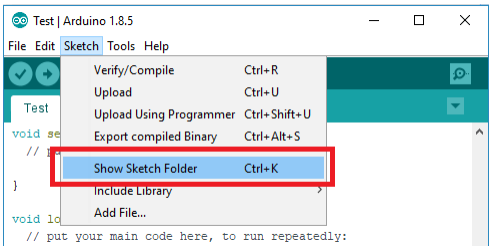
Uploading Files using the Filesystem Uploader

To upload files to the ESP32 filesystem follow the next instructions.

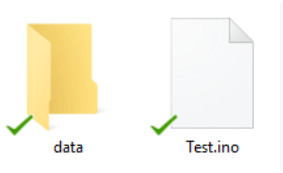
1) Create an Arduino sketch and save it.

2) Then, open the sketch folder.

You can go to Sketch > Show Sketch Folder.

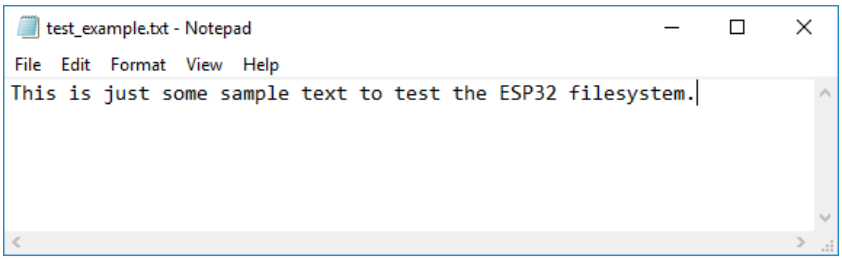


3) Inside that folder, create a new folder called data.



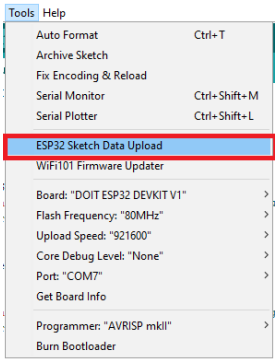
4) Inside the data folder is where you should put the files you want to be saved into the ESP32 filesystem.

As an example, create a .txt file with some text called test_example.



5) Then, to upload the files, in the Arduino IDE, you just need to go to

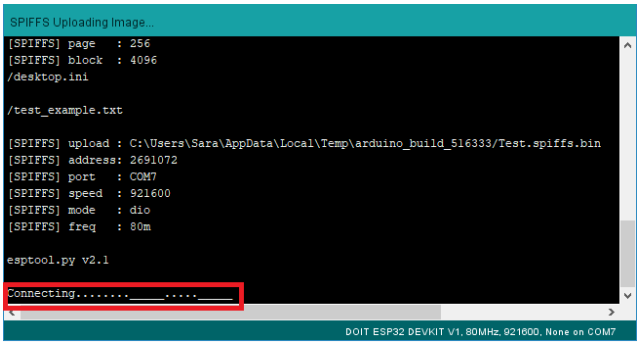
Tools > ESP32 Sketch Data Upload



****** Note:***

in some ESP32 development boards you need to keep the ESP32 on-board "BOOT" button pressed while it's uploading the files.

When you see the "Connecting …….____……" message, you need to press the ESP32 on-board "BOOT" button.



When you see the message “SPIFFS Image Uploaded“,

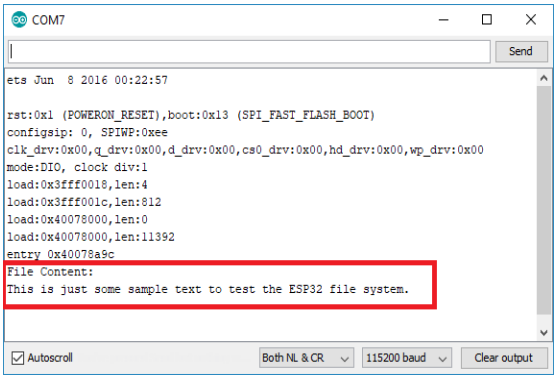
the files were successfully uploaded to the ESP32 filesystem.

Testing the Uploader

Now, let's just check if the file was actually saved into the ESP32 filesystem.

Simply upload the following code to your ESP32 board.

```
/*********
  Chris Barden - Dustin Engle
  Complete project details at https://itshere.io
*********/

#include "SPIFFS.h"

void setup() {
  Serial.begin(115200);

  if(!SPIFFS.begin(true)){
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }

  File file = SPIFFS.open("/test_example.txt");
  if(!file){
    Serial.println("Failed to open file for reading");
    return;
  }

  Serial.println("File Content:");
  while(file.available()){
    Serial.write(file.read());
  }
  file.close();
}

void loop() {

}
```



After uploading,

open the Serial Monitor at a baud rate of 115200.

Press the ESP32 "ENABLE" button.

It should print the content of your .txt file on the Serial Monitor.

You've successfully uploaded files to the ESP32 filesystem using the plugin.

Introduction to LoRa

LoRa or Long Range is a digital wireless data communication protocol developed by Cycleo of Grenoble, France. Semtech, a leading supplier of supplier of analog and mixed-signal integrated circuits, and the parent company of Cycleo, owns the IP for LoRa transmission technique.

LoRa is a long-range wireless communication protocol that transmits over sub-gigahertz radio frequency bands such as 169 MHz, 433 MHz, 868 MHz and 915 MHz.

Very long range transmission of over 10 km in rural areas are enabled by LoRa even at a low power consumption.

What kind of Technology sits under the LoRa covers ?

The physical layer is LoRa upon which a communication protocol is built. The communication layer can be Long Range Wide Area Network (LoRaWAN) or Symphony Link.

LoRaWAN is a networking protocol designed to wirelessly connect battery-operated devices to the internet across global national or regional networks. It meets IoT requirements such as bi-directional communication, mobility, end-to-end security and localization services.

LoRaWAN is defined by the LoRa Alliance, a non-profit consortium of over 500 member companies whose objective is to enable the large scale deployment of Low Power Wide Area Networks (LPWAN) by developing and promoting the LoRaWAN open standard.

Technology suppliers forming the alliance include Cisco, IBM, Actility and IMST to name some, while telecoms members include Bouygues Telecom, SingRel, Swisscom, KPN, FastNet and Proximus.

Symphony Link is an open source wireless system built by Link Labs. It is used by enterprise and industrial customers who appreciate the range of LoRa but require high reliability and advanced features in their LPWA system.

Core Features of LoRa

LoRa is a spread spectrum modulation technique, which indicates a method by which an electrical, electromagnetic or acoustic signal generated with a certain bandwidth is deliberately spread in frequency domain, resulting in a signal with a wider bandwidth. It is derived from chirp spread spectrum (CSS) technology, a spread spectrum technique that uses wideband linear frequency modulated chirp pulses to encode information.

LoRa allows the amount of spread used to be selected, which helps determines the data rate and influences the sensitivity of a radio.

To address interference, the communication technology uses Forward Error Correction Coding (FEC). This is a technique employed to control errors in data transmission over noisy or unreliable communication channels. The sender encodes the message in a redundant way by using error-correcting code (ECC).

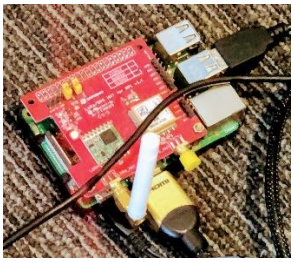
Within the sub-GHz spectrum, LoRa chips can transmit at a variety of frequencies and data rates. This capability helps the gateway in adapting to changing conditions and optimizing data exchange with each device.

Network topology of LoRa and LoRaWAN

LoRaWAN network architecture is deployed as a star-of-stars topology. The gateways relay messages between end devices and a central network server.

The gateways are connected to the network server via standard IP connections and serve as a transparent bridge converting RF packets to IP packets and vice versa.

End Nodes and Gateways



The end nodes of the architecture are LoRa embedded sensors.

The nodes have sensors that detect changing temperature, humidity, GPS and other parameters; a LoRa transponder to emit signals over LoRa patented radio transmission methods; and may optionally feature a micro-controller with on-board Memory.

The sensors may either connect to the LoRa transponder chip or be an integrated unit with the LoRa transponder chip embedded. The microcontrollers can be programmed in micro-Javascript or micro-Python. It enables developers to use the data from sensors for specific use cases.

The LoRaWAN end nodes usually employ Low Power and Class A or Class B batteries. Battery-driven LoRa embedded sensors can last anywhere from two to five years. The sensors can transmit signals over distances of 1km to 10km.

We are not using LoRaWAN for this smart lock kit.

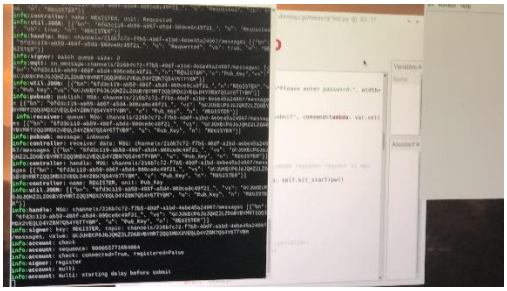
LoRa sensors transit data to LoRa gateways, which connect to the internet via the standard IP protocol and convey the data from LoRa embedded sensors to the internet, which can be a server, network or cloud.

Perpetually connected to a power source, the Gateway devices connect to the network server via standard IP connections, and as mentioned previously, do the job of converting RF packets to IP packets and vice-versa.

LoRa Gateways need to operate over open frequency in order for them to receive data from any sensor the immediate surrounding. For this reason, LoRa Gateways lack the ability to decrypt sensor data.

As far as home applications are concerned, LoRa offers immense potential to deliver home automation for IoT-enabled smart appliances as well as enhance home security.

How to set up and configure the Smart MailBox Lock system :



1. Connect the following to the Raspberry Pi
   (in order)
   a. Mouse and Keyboard (USB)
   b. Monitor (HDMI)
   c. Power supply (Micro USB)
   d. After OS boot up, connect to WIFI

2. Map to the following file directory:

   home > pi > Code/smart-mailbox > gateway

   a. Show hidden files
      right-click file directly > Show hidden files

   b. Make the following modifications to the files below:

      i. Modify (.env)

         (Open with TextEditor)

         remove all values from the following

         1. KIT_CHANNEL
         2. KIT_DEVICE_ID
         3. KIT_DEVICE_KEY
         4. KIT_REGISTERED
         5. GATE_PUB_KEYS
         6. Click Save

ii. Modify (gateseed.dat)
(Open with Genie)

1. remove all contents

2. Click Save

iii. Remove secret.dat

1. delete file

3. Download Mobile App:
https://play.google.com/store/apps/details?id=io.itshere.mobile

a. On Mobile App
register as New User
(Need a create an account? Click on Register)

b. Login to Mobile App using (email and password) created on register page

4. Setup Mailbox

a. Make sure to remove ***OLD gateways*** on Mobile App (if they exist already)

b. From Raspberry Pi Monitor

i. Map to
home > pi > Code/smart-mailbox > gateway

ii. Right-click on gateway
and select (open from command line window)

c. From command line window

i. Execute the following script:
python qrTest3.py

1. Wait for the following before proceeding:
info:signer: batch queue size:0

ii. QR Code will appear with Dialog box saying Gateway has not been activated

d. From Mobile App:
Go to Gateway module

i. +Add Gateway

ii. Enter Name: <Any name you want>

iii. Scan QR code (from Pop-up dialog on Pi Monitor)

iv. Click Submit on Mobile App

e. From Raspberry Pi Monitor

i. Wait for the following before proceeding:
info:signer: batch queue size:0

ii. Click OK on QR code dialog box (Gateway is now activated)

iii. Password pop-up will appear

1. enter password (e.g. 12345)

2. confirm password (e.g. 12345)

iv. Wait for the following before proceeding:
info:signer: batch queue size:0

v. Dialogbox will appear saying:
Please use App to scan QR code on Mailbox
do not click continue until code is scanned

5. Setup Mailbox

a. From Mobile App:
Go to Gateway > Click 'Add Mailbox'

i. Enter Name: <Any name you want>

b. From Mailbox LOCK:

i. Click Reset button on lock
QR Code should appear

c. From Mobile App:

i. Scan QR code on Mailbox lock

ii. Click Submit on Mobile App

d. From Mailbox LOCK:

i. Click Zero (prd) on Mailbox lock

e. From Raspberry Pi Monitor

i. Wait for the following before proceeding:
info:signer: batch queue size:0

ii. Click continue on pop-up for Mailbox

1. Should see the following :
Mailbox 1: REGISTSRED (confirmation)