

# SparkFun LoRa Gateway 1-Channel Hookup Guide

## Introduction

**Note:** Please note that this tutorial is for WRL-15006. The tutorial is applicable for WRL-18074. The only difference is between the two SKUs is that the the ESP32-WROOM-32 module was updated from 4MB to 16MB.

If you are using this with the older version [SPX-14893], please refer to the ESP32 LoRa 1-CH Gateway, LoRaWAN, and the Things Network tutorial.

So you've designed an automatic shepherding robot but you still have to go out to the field to make sure it is working? Worry no more, you can use long-range radio to keep tabs on that 'bot through the internet of things! All you need is an interpreter to speak the language, and the LoRa Gateway 1-Channel does just that.



SparkFun LoRa Gateway - 1-Channel (ESP32)  
○ WRL-18074

The LoRa Gateway 1-Channel is a monster 3-network capable device thanks to an ESP32 module and a RFM95W LoRa modem. The RFM95W handles the 915 MHz band while the ESP32 takes care of bluetooth and WiFi. One of the ideal uses is to convert LoRa (Long Range) radio messages into data packets that you can access via the web, but of course the flexibility it offers can be put to many more uses!

This guide will go over the hardware on the board, how to program it in Arduino, how to create a single channel LoRa gateway, and finally how to create a LoRa device on The Things Network.

## Required Materials

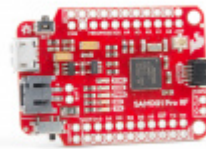
The LoRa Gateway 1-Channel can act as either a gateway or a device, but not both at the same time. To really be sure that your setup works as expected you should have another LoRa device to listen to, and/or another LoRa gateway to transmit to. The good news is that the LoRa Gateway 1-Channel can act as both so if you have two then you're all set.

If you only have one LoRa 1-Channel gateway then you can choose one of these products to test it:



SparkFun LoRa Gateway - 1-Channel (ESP32)

○ WRL-18074



SparkFun Pro RF - LoRa, 915MHz (SAMD21)

○ WRL-15836



LoRa Raspberry Pi Gateway with Enclosure

○ WRL-15336



LoRa Raspberry Pi 4 Gateway with Enclosure

○ WRL-16447

To program the LoRa Gateway 1-Channel you will need a micro-B USB cable and a computer with the Arduino IDE installed. If you want to make a permanent installation away from your computer then consider powering it with a USB wall adapter or USB battery pack.

## Tools

To use the 915 MHz radio on the gateway you will need an antenna - for which you have two choices. You may cut a length of solid-core wire to approx 3" or use a 915 MHz antenna with a U.FL connector. If you choose the wire route then you will also need a soldering iron and tools to attach your antenna to the board.



Interface Cable RP-SMA to U.FL  
● WRL-00662



Weller WLC100 Soldering Station  
○ TOL-14228



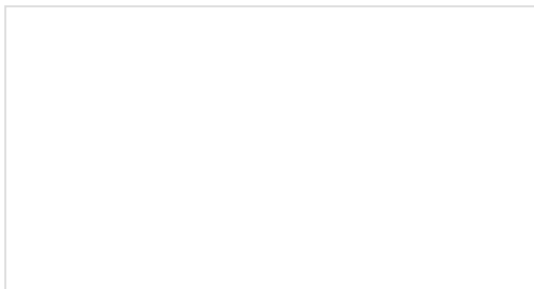
Pycom LoRa and Sigfox Antenna Kit - 915MHz  
● WRL-14676



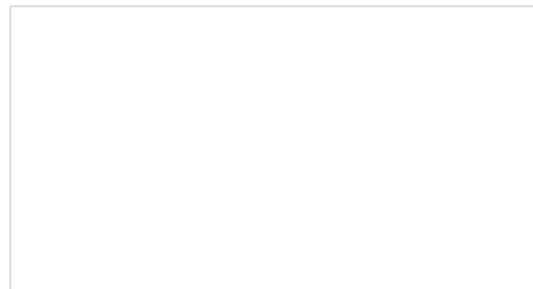
915MHz LoRa Antenna RP-SMA - 1/4 Wave 2dBi  
● WRL-14875

## Suggested Reading

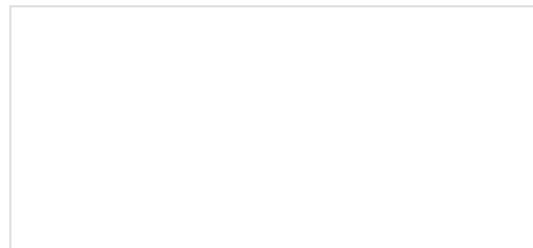
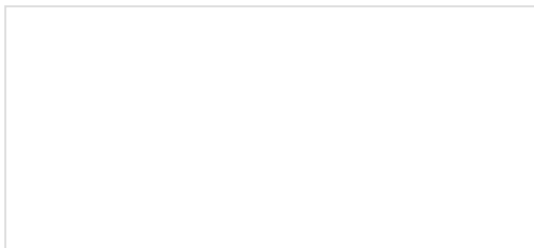
If you aren't familiar with the following concepts, we recommend checking out these tutorials before continuing.



**How to Solder: Through-Hole Soldering**  
This tutorial covers everything you need to know about through-hole soldering.



**Installing Arduino IDE**  
A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.



## Serial Terminal Basics

This tutorial will show you how to communicate with your serial devices using a variety of terminal emulator applications.

SparkFun Serial Basic CH340C Hookup Guide  
SparkFun Serial Basic Breakout takes advantage of USB-C and is an easy-to-use USB-to-Serial adapter based on the CH340C IC from WCH. With USB-C you can get up to three times the power delivery over the previous USB generation and has the convenient feature of being reversible.

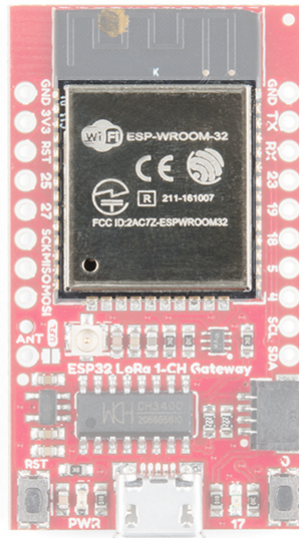
## Hardware Overview

The LoRa Gateway 1-Channel is chock full of functionality:

- ESP32-WROOM-32 module
- WiFi, BT+BLE microcontroller
- Integrated PCB antenna
- Hope RFM95W LoRa modem
- Frequency range: 868/915 MHz
- Spread factor: 6-12
- SPI control interface
- U.FL antenna connector for LoRa radio
- Reset and ESP32 pin0 buttons
- 14 GPIO ESP32 pin-breakouts
- Power and user LEDs
- Qwiic connector
- CH340C USB-to-Serial interface
- Micro-B USB connector for power and programming
- Voltage input range: **3.3V-6V** max

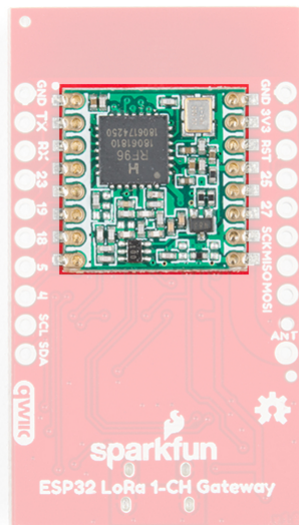
### ESP32

The brains of the gateway is an ESP32-WROOM-32 module, shown below. It has all the same features as the SparkFun ESP32 Thing rolled up into one sweet little package. WiFi, Bluetooth, 240 MHz processing speed, and a bunch of I/O pins make it a great foundation for the gateway.



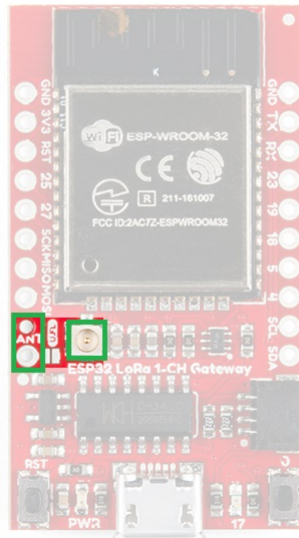
## RFM95W

Every LoRa gateway needs to speak the Chirp Spread Spectrum (CSS) radio language and the RFM95W module does just that in the 915 MHz centered ISM band. The limitation on this device is it can only listen to one LoRa channel at a time, unlike full multi-channel LoRa gateways.



## Antenna Connections

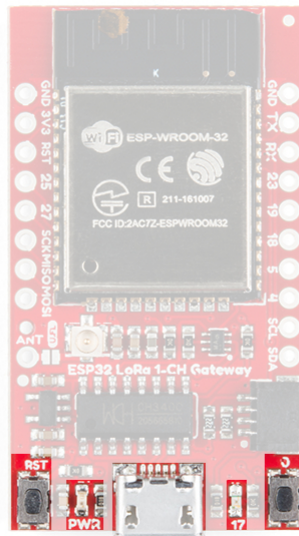
The LoRa Gateway 1-Channel sports both a through-hole antenna connection with strain relief as well as a U.FL connection for higher performance antennas.



**Note:** The LoRa Gateway 1-Channel ships with the u.FI jumper open. Users will need to close the jumper to use the u.FI connector. Check out this tutorial for tips on working with solder jumpers.

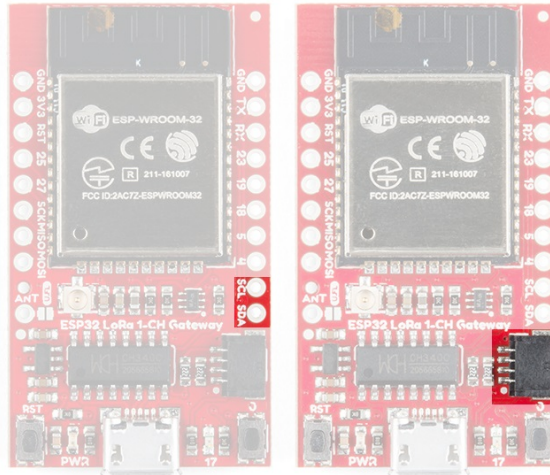
## User Buttons, LEDs, and USB

On the bottom left side of the board you will find the power led and the reset button. Opposing are a button connected to pin 0 and an led connected to pin 17. The buttons exist to force the ESP32 into programming mode in case the automatic process by the CH340C USB-serial bridge chip fails, but after that you can use button 0 as an active-low input for your sketch. The ESP also uses the USB-serial bridge as the default serial port so there's nothing else you need to connect to your computer.



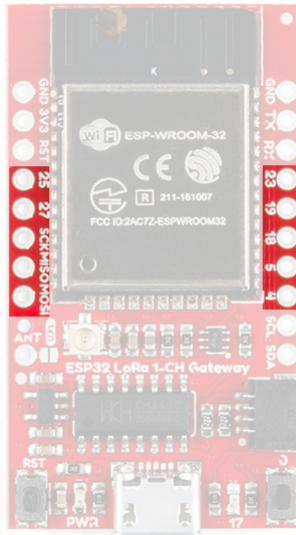
## Qwiic Connector

The default I<sup>2</sup>C lines of the ESP32 are broken out to both PTH pads and the super-convenient Qwiic connector. This means you can easily add peripherals to your gateway or sensors to your LoRa device!



## IO Pins

For anything other than I<sup>2</sup>C you can use the SPI lines and/or 7 GPIO pins which are all broken out to PTH pads around the board.



## Programming the ESP32 With Arduino

**Note:** This example assumes you are using the latest version of the Arduino IDE on your desktop. If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE. If you have not previously installed an Arduino library, please check out our installation guide.

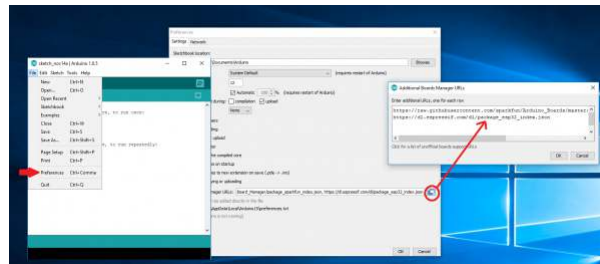
We will be using the Arduino IDE to upload new code to the Gateway. To get everything working the way it oughtta you'll have to install the ESP32 Arduino core - a set of tools and code that translates Arduino code to something that the ESP32 understands. You can also make life a little bit easier by installing a custom board definition for the LoRa Gateway 1-Channel.

### Install ESP32 Arduino Core

The ESP32's relationship with Arduino is growing and now it is very easy to install the core - the Arduino IDE can handle it nearly on its own. All you need to do is make sure you have Arduino IDE version 1.8 or later, then paste

```
https://dl.espressif.com/dl/package_esp32_index.json
```

into the *Additional Board Manager URLs* field of the preferences window, like this:



*Hard time seeing? Click on the image for a closer look!*

Now accept the changes and restart the Arduino IDE. Next open the **Board Manager** from the top of **Tools > Board** and search for ESP32. Click "Install" on the search result, after a little while the text besides the name should change to "Installed." Re-start the IDE for good measure.

### Install the LoRa Gateway 1-Channel Board Definition

Download the variant file zip folder and extract it into the location of your ESP32 Arduino core installation. If you used the board manager as shown above then the file path is likely very similar the following:

#### Windows:

- `C:\Users\\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.0\variants.`

#### Mac:

- `/Users/<user.name>/Library/Arduino15/packages/esp32/hardware/esp32/1.0.0/variants`

If you see other variant folders in the folder listed above, it's a good indication that you're in the right place.

### DOWNLOAD THE SPARKFUN LORA GATEWAY 1-CHANNEL ESP32 VARIANT DEFINITION (ZIP)

**Note:** If you cannot find your "AppData" folder, it may be hidden. For windows, click on the "View" tab of your explorer window and check the "Hidden Items" checkbox as seen here:



To make sure that Arduino knows to look for the variant that you just installed you will need to paste the following text into the `boards.txt` file in the `~esp32\1.0.0` folder. Technically you can paste the text in anywhere but to maintain readability, do it before a long string of `#`'s. I prefer to put my definitions at the top so that they appear first in the list of boards in the boards manager.



#####

sparkfun\_lora\_gateway\_1-channel.name=SparkFun LoRa Gateway 1-Channel

sparkfun\_lora\_gateway\_1-channel.upload.tool=esptool\_py  
sparkfun\_lora\_gateway\_1-channel.upload.maximum\_size=1310720  
sparkfun\_lora\_gateway\_1-channel.upload.maximum\_data\_size=294912  
sparkfun\_lora\_gateway\_1-channel.upload.wait\_for\_upload\_port=true

sparkfun\_lora\_gateway\_1-channel.serial.disabledDTR=true  
sparkfun\_lora\_gateway\_1-channel.serial.disableRTS=true

sparkfun\_lora\_gateway\_1-channel.build.mcu=esp32  
sparkfun\_lora\_gateway\_1-channel.build.core=esp32  
sparkfun\_lora\_gateway\_1-channel.build.variant=sparkfun\_lora\_gateway\_1-channel  
sparkfun\_lora\_gateway\_1-channel.build.board=ESP32\_DEV

sparkfun\_lora\_gateway\_1-channel.build.f\_cpu=24000000L  
sparkfun\_lora\_gateway\_1-channel.build.flash\_size=4MB  
sparkfun\_lora\_gateway\_1-channel.build.flash\_freq=40m  
sparkfun\_lora\_gateway\_1-channel.build.flash\_mode=dio  
sparkfun\_lora\_gateway\_1-channel.build.boot=dio  
sparkfun\_lora\_gateway\_1-channel.build.partitions=default

sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.default=Default  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.default.build.partitions=default  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.minimal=Minimal (2MB FLASH)  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.minimal.build.partitions=minimal  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.no\_ota=No OTA (Large APP)  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.no\_ota.build.partitions=no\_ota  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.no\_ota.upload.maximum\_size=2097152  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.min\_spiffs=Minimal SPIFFS (Large APPS with OTA)  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.min\_spiffs.build.partitions=min\_spiffs  
sparkfun\_lora\_gateway\_1-channel.menu.PartitionScheme.min\_spiffs.upload.maximum\_size=1966080

sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.qio=QIO  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.qio.build.flash\_mode=dio  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.qio.build.boot=qio  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.dio=DIO  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.dio.build.flash\_mode=dio  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.dio.build.boot=dio  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.qout=QOUT  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.qout.build.flash\_mode=dout  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.qout.build.boot=qout  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.dout=DOUT  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.dout.build.flash\_mode=dout  
sparkfun\_lora\_gateway\_1-channel.menu.FlashMode.dout.build.boot=dout

sparkfun\_lora\_gateway\_1-channel.menu.FlashFreq.80=80MHz  
sparkfun\_lora\_gateway\_1-channel.menu.FlashFreq.80.build.flash\_freq=80m  
sparkfun\_lora\_gateway\_1-channel.menu.FlashFreq.40=40MHz  
sparkfun\_lora\_gateway\_1-channel.menu.FlashFreq.40.build.flash\_freq=40m

```
sparkfun_lora_gateway_1-channel.menu.FlashSize.4M=4MB (32Mb)
sparkfun_lora_gateway_1-channel.menu.FlashSize.4M.build.flash_size=4MB

sparkfun_lora_gateway_1-channel.menu.UploadSpeed.921600=921600
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.921600.upload.speed=921600
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.115200=115200
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.115200.upload.speed=115200
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.256000.windows=256000
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.256000.upload.speed=256000
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.230400.windows.upload.speed=256000
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.230400=230400
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.230400.upload.speed=230400
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.460800.linux=460800
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.460800.macosx=460800
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.460800.upload.speed=460800
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.512000.windows=512000
sparkfun_lora_gateway_1-channel.menu.UploadSpeed.512000.upload.speed=512000
```

## Upload Blink

To make sure everything is hunkey-dorey let's blink the LED. Make sure the correct board is selected (SparkFun LoRa Gateway 1-Channel if you installed the variant like we did above) and select the proper programming port. Then open the "Blink" example and change the led pin definition to "LED\_BUILTIN" (or pin 17 if you don't have the variant installed). If you hit upload the code should compile and be transferred to the board, and the pin 17 led should begin to blink.

Now that you are in control of the ESP32 we can move on to exciting things! The remaining portions of this guide will focus on sending a "Hello world!" message from a LoRa device to the internet.

## LoRaWAN Roadmap

LoRa, LoRaWAN, and the Internet of Things is a very big topic. Before we go and get lost in code let's make sure we know what the end goal is and roughly how we plan to get there.

Let's start with the Internet of Things (IoT). Really all IoT is is the idea that we can add inter-connectivity to a large portion of the things we interact with on a day-to-day basis. For example, if your refrigerator kept track of what was inside and could talk to your cell phone, then when you were at the store you wouldn't be left wondering if you need another case of kombucha to pair with the 6 bags of quinoa chips you just bought. So the Internet of Things is about connectivity.

Connectivity is pretty well-solved in the home with WiFi and Bluetooth, but what if your refrigerator was in the middle of a field without access to the internet? (Okay bad example, but the refrigerator is just a stand-in for *your* connected device). This is where LoRa comes in. LoRa is "Long Range" radio which was designed for low power consumption and long range transmissions at the expense of bandwidth. This means you can send a little bit of data a long way.

Okay, so now your field refrigerator can send messages a long way -- ideally far enough to reach to the nearest Starbucks - or any other place that has WiFi. Of course nobody at the coffee shop knows about your far-distant fridge (and certainly the WiFi modems won't be listening for messages) so you need something that is dedicated to bridge from LoRa messages to internet traffic - called a "gateway." As long as you have something that can speak both LoRa and "Internet" then you could make your own solution, but there is a easier and better option. This is where LoRaWAN comes in.

LoRaWAN is a public specification for the system that would be at Starbucks listening for messages from the fridge. Two important benefits of LoRaWAN are that your data is encrypted during transmission and that your fridge could get up and walk to the next state over (again - just a metaphor!) and the messages could still be picked up by a gateway that someone else had built. Since the messages are secure that person won't know about your stinky cheese but the message will still get back to you over the internet. Groups like The Things Network organize everyone's efforts to make this possible.

The goal of this guide is to walk through that whole process and get a "Hello World!" message from a remote device, into a gateway, and onto the internet. We will go a little bit out-of-order because it makes things easier. First we will create the gateway, then we will fire up a device to send the data, and finally we will create an internet application to look for the data.

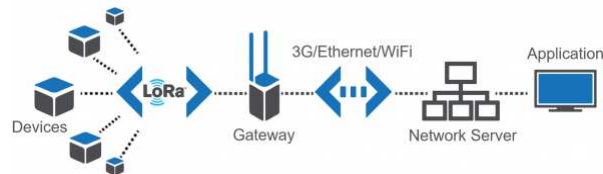


Image courtesy of Jensd. If you want to read more about what LoRa and LoRaWAN are check out this great post on Jensd's I/O Buffer.

## Your Device

A "device" is the remote system that is sending (and in some cases receiving) data. We will set up a device to actually send the "Hello World!" message in the section "Turning a Gateway into A Device."

## Your Gateway

The name of the ESP LoRa Gateway 1-Channel is a dead giveaway. With all its glorious wireless connectivity it acts as the bridge that speaks both WiFi and LoRa. The section "Single-Channel LoRaWAN Gateway" will cover all the steps you need to make this part.

## Your Application

An "application" is the software that you set up to look for and handle the messages that make it to the internet. We will create an application using The Things Network that will be on the lookout for your "Hello World!" message.

# Single-Channel LoRaWAN Gateway

## Making a LoRa Gateway

Thanks to 915 MHz LoRa AND WiFi connectivity, the LoRa Gateway 1-Channel really shines as an inexpensive gateway in a LoRaWAN network. This section will show you how to make your own gateway and access it on the internet.

If you've followed along with this hookup guide then you should already be able to program the LoRa Gateway 1-Channel. The next step is to download a library to run LoRaWAN and modify it to suit our board and needs.

## Download the Library

The ESP 1-ch Gateway code is hosted on GitHub by things4u. You can get the latest and greatest there, or download our archived copy here:

## ARCHIVED ESP-1CH-GATEWAY-V5.0 (ZIP)

This repository includes both the Arduino sketch and the libraries it depends on. Before compiling the sketch you'll need to extract all libraries from the repository's "library" folder into your Arduino sketchbook's "libraries" folder. For more help installing the libraries, check out the Getting Started section of the README.

To open the example code, open the **ESP-sc-gway.ino** file. When the IDE loads, it should include another dozen-or-so tabs -- it's a hefty, but well-segmented sketch!

### Configure the Gateway Sketch

Before uploading the ESP-1ch-Gateway sketch to your board, you'll need to make a handful of modifications to a couple of files. (Use Ctrl-F to search the file for the setting you want to modify) Here's a quick overview:

#### ESP-sc-gway.h

This file is the main source of configuration for the gateway sketch. The definitions you'll probably have to modify are:

- **Radio**
  - `_LFREQ` -- This sets the frequency range your radio will communicate on. Set this to either 433 (Asia), 868 (EU), or 915 (US)
  - `_SPREADING` -- This sets the LoRa spread factor. SF7, SF8, SF9, SF10, SF11, or SF12 can be used. Note that this will affect which devices your gateway can communicate with.
  - `_CAD` -- Channel Activity Detection. If enabled (set to 1) CAD will allow the gateway to monitor messages sent at any spread factor. The tradeoff if enabled: very weak signals may not be picked up by the radio.
- **Hardware**
  - `OLED` -- This board does not include an OLED, **set this to 0**.
  - `_PIN_OUT` -- This configures the SPI and other hardware settings. **Set this to 6**, we'll add a custom hardware definition later.
  - `CFG_sx1276_radio` -- Ensure this is defined and `CFG_sx1272_radio` is *not*. This configures the LoRa radio connected to the ESP32.
- **The Things Network (TTN)**
  - `_TTNSERVER` -- The TTN router that your gateway will hand its data over to. **router.eu.thethings.network recommended**. See the note below.
  - `_TTNPORT` -- 1700 is the standard port for TTN
  - `_DESCRIPTION` -- Customize the name of your gateway
  - `_EMAIL` -- Your email address, or that of the owner of the gateway
  - `_LAT` and `_LON` -- GPS coordinates of your gateway
- **WiFi**
  - Add at least one WiFi network to the `wpa wpa[]` array, but leave the first entry blank. For example:

```
wpa wpa[] = {
  { "", "" }, // Reserved for WiFi Manager
  { "my_wifi_network", "my_wifi_password" }
};
```

**Note:** The Things Network doesn't allow single-channel gateways to route data through the 'regular' routers (e.g. 'eu.thethings.network') so you need to use the 'development' router equivalents

```
'router._your_chosen_router_here_'. We've also found that the best router to choose is the EU router
**router.eu.thethings.network**.
```

There are a lot of other values which can all optionally be configured. For a complete rundown, check out the Editing the ESP-sc-gway.h part of the README.

## loramodem.h

This file defines how the LoRa modem is configured, including which frequency channels it can use and which pins the ESP32 uses to communicate with it. Be careful modifying most of the definitions in here, but one section you will have to modify is the `_PIN_OUT` declarations.

First, find the line that says `#error "Pin Definitions _PIN_OUT must be 1(HALLARD) or 2 (COMRESULT)"` and **delete it**. Then copy and paste these lines in its place (between the `#else` and `#endif`):

```
struct pins {
  uint8_t dio0 = 26;
  uint8_t dio1 = 33;
  uint8_t dio2 = 32;
  uint8_t ss = 16;
  uint8_t rst = 27; // Reset not used
} pins;
#define SCK 14
#define MISO 12
#define MOSI 13
#define SS 16
#define DI00 26
```

The `int freqs[]` array can be adjusted, if you want to use different subbands, but, beyond that, there's not much else in here we recommend modifying.

## Upload the Sketch

With your modifications made, try compiling and uploading the sketch to your ESP32. After it's uploaded, open up your serial monitor and set the baud rate to **115200**. Debug messages here can be very handy, and finding your gateway's IP address is critical if you want to monitor the web server.

The sketch may take a long time to set up the first time through -- it will format your SPIFFS file system and create a non-volatile configuration file. Once that's complete, you should see the ESP32 attempt to connect to your WiFi network, then initialize the radio.

After the ESP32 has connected, look for it to print out an IP address. Open up your computer's web browser and plug that into the address bar. You should be greeted by the ESP Gateway Config web portal:

The screenshot shows a web browser window titled "ESP Gateway Config" with the URL "10.8.255.234". The page contains three main sections:

### Package Statistics

Counter	C 0	C 1	C 2	Flags	Package
Packages Downlink				0	
Packages Uplink Total				4	0%
Packages Uplink OK				2	
SP revd	2	0	0	2	50%
SP8 revd	2	0	0	2	50%
SP9 revd	0	0	0	0	0%
SP10 revd	0	0	0	0	0%
SP11 revd	0	0	0	0	0%
SP12 revd	0	0	0	0	0%

### Message History

Time	Node	C	Freq	SP	RSSI
Wednesday 1-8-2018 18:36:34	26 02 11 84	0	865700000	7	-22
Wednesday 1-8-2018 18:36:28	26 02 11 84	0	865700000	7	-22
Wednesday 1-8-2018 18:25:11	26 02 16 76	0	865700000	8	-107
Wednesday 1-8-2018 18:23:11	26 02 16 76	0	865700000	8	-107

### Gateway Settings

Setting	Value	Set
CAD	ON	ON OFF
RFIF	OFF	ON OFF
SP Setting	AUTO	ON OFF
Channel	0	- +
Debug level	2	- +
Debug port	COM	CAD 8K 7K
USB Debug	PRE	MAN SCL SDO
Francometer Internal Sensor	2	0 502

Config and log page served by the ESP32.

This web page can be used to monitor messages coming through and what frequencies and spread factors they came in on. It can also be used to change your gateway's configuration on-the-fly. You can adjust the channel or spread factor, or turn CAD on/off, or even turn on simplistic frequency-hopping.

Of course, to see any messages get through you'll need a LoRa device (or *devices*) set up to communicate with your gateway. Check out the next section for a quick guide on setting up a second ESP32 LoRa board as a LoRa device. Alternately if you have a different SparkFun LoRa-enabled product like the SAMD21 Pro RF follow that product's guide to set up a LoRaWAN device.

## Troubleshooting

If you copy your IP address into the web browser and get a message saying the website cannot be reached, make sure your LoRaWAN and your computer are connected to the same network.

## Turning a Gateway Into a Device

If you have a pair of ESP32 gateway's you can use one of them as a LoRaWAN device. This section will get you set up with our preferred LoRaWAN Arduino library and a simple example sketch to get started.

### Get the Arduino-LMIC Library to Set Up a Device

If you have a pair of ESP32 Gateways and want to set one of them up as a LoRa device, we recommend using the Arduino-LMIC library. There seem to be many versions of the Arduino-LMIC library hanging around, we've had good success with the library forked by mcci-catena. If you don't want to use GitHub you can download an archived .zip file here. It will work for this guide but it may not be completely up-to-date.

**ARCHIVED ARDUINO-LMIC FROM MCCI-CATENA (ZIP)**

To install the library, download the ZIP file and use the Arduino's "Add ZIP library" feature (**Sketch > Include Library > Add .ZIP Library**) to import the zip file into your Arduino sketchbook.

### Configure LMIC

Before can use an example sketch in the LMIC library, you'll need to configure it. To configure the library, navigate to the "arduino-lmic" library in your **..{Arduino Sketchbook}/libraries** folder. Then go to **project\_config** and open **lmic\_project\_config.h**.

Here you'll define which frequency bands your LoRa device will use. You can also turn on debugging and enable/disable a host of features with definitions in this file.

Make sure you uncomment one (and only one!) of the "CFG..." declarations. If you're in the USA, uncomment `#define CFG_us915`. Ultimately, this frequency should match what you set in the gateway.

Below is my config file -- I've turned on debugging, because I'm a log junkie.

```
// project-specific definitions
//#define CFG_eu868 1
#define CFG_us915 1
//#define CFG_au921 1
//#define CFG_as923 1
// #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP /* for as923-JP */
//#define CFG_in866 1
#define CFG_sx1276_radio 1

//#define LMIC_PRINTF_TO Serial
#define LMIC_DEBUG_LEVEL 2

//#define DISABLE_INVERT_IQ_ON_RX
```

If you try to use the "raw" example in this library, you'll need to uncomment `DISABLE_INVERT_IQ_ON_RX`, otherwise keep it commented-out.

### Modify the "SPI.begin" call

Just to be sure your pin definitions are correct, I recommend modifying the `SPI.begin` line in **arduino-lmic/src/hal/hal.cpp** (line 136) to:

```
SPI.begin(14, 12, 13, 16);
```

This will ensure that your SPI pins are set correctly -- this may only be necessary if you're not using the custom "SparkFun LoRa gateway 1-Channel" Arduino board.

### Load the Single-Channel Device Example

We've taken one of the examples in the Arduino-LMIC library and modified it to work more reliably with a single-channel gateway.

Click the button below to download the example. Then open up **ESP-1CH-TTN-Device-ABP.ino**:

[DOWNLOAD THE SINGLE-CHANNEL LORAWAN DEVICE EXAMPLE \(ZIP\)](#)

Among the modifications in this example are the pin map between radio and ESP32 -- set with these lines:

```
const lmic_pinmap lmic_pins = {
  .nss = 16,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 5,
  .dio = {26, 33, 32},
};
```

We've also modified the enabled channels to only use a single channel with the lines below (note this modification hasn't yet been tested on non-US frequency bands).

```
// First disable all sub-bands
for (int b = 0; b < 8; ++b) {
    LMIC_disableSubBand(b);
}
// Then enable the channel(s) you want to use
LMIC_enableChannel(8); // 903.9 MHz
```

The spread factor is set near the end of `setup()` with the `LMIC_setDrTxpow(DR_SF7, 14);` line. This can be replaced with `DR_SF8`, `DR_SF9`, or `DR_SF10` at the default 903.9MHz frequency.

You can't compile successfully until you fill in these lines:

```
// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWKSKEY[16] = { PASTE_NWKSKEY_KEY_HERE };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM APPSKEY[16] = { PASTE_APPSKEY_KEY_HERE };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0xPASTE_DEV_ADDR_HERE;
```

with *something*. If you want to test the gateway right now go ahead and fill the two arrays with 16 bytes each and the `DEVADDR` with a 4-byte wide number. If you do this and upload the code then you should see the gateway receiving messages. (If you're having trouble make sure that the device and the gateway are configured to use the same channel and spreading factor).

If you are OK waiting then leave these blank until you get your actual keys and device address from The Things Network in the next section.

## Routing Into The Things Network

The final components to a LoRaWAN network are a server and application. Although it is possible to make your own server we've already configured the gateway to send data to The Things Network through one of their routers like "router.eu.thethings.network". This is an easy and free way to get started with LoRaWAN so we recommend starting this way.

### Single-Channel Blues

At the trade-off of being low-cost, this gateway is only capable of monitoring a single LoRa channel on a limited set of spread factors. Single-channel gateway's don't get much support from LoRaWAN platforms like The Things Network, as they are not, necessarily, LoRaWAN-compliant. They are, however, a great way to begin exploring the world of LoRa and LoRaWAN!

If you haven't already, head to [thethingsnetwork.org](https://thethingsnetwork.org) and create an account. Once that's done, go to your **Console**.

**THE THINGS NETWORK CONSOLE**

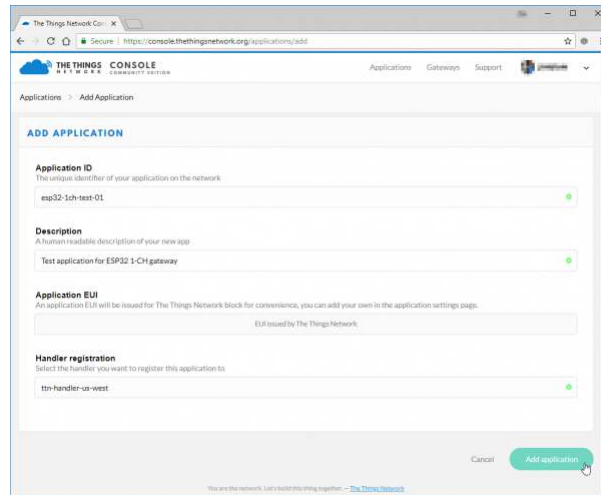


## Create an Application

If your data is being sent to The Things Network's servers then you can create an application to access that data. This works by registering devices with unique cryptographic keys so that your data is only visible to you.

In order to register a device, you first need to create an application to house it under. In the console, click "**Applications**" then click "**add application**".

Fill out any ID and description you'd like. The **Application EUI** will automatically be generated when you create the application. You can also pick your preferred handler for the application (e.g. ttn-handler-us-west).

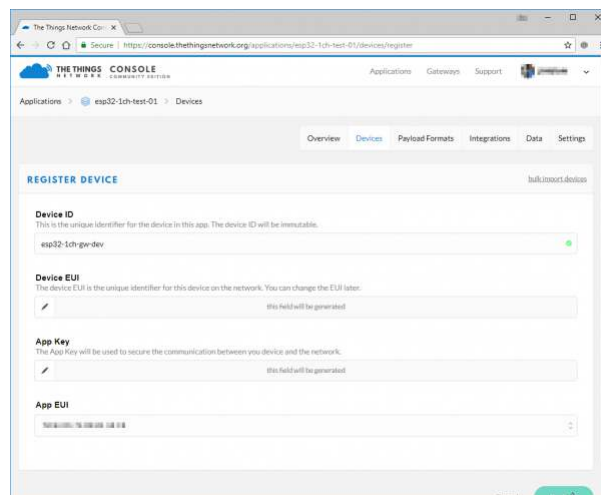
A screenshot of the The Things Network Console showing the 'Add Application' form. The form has four main sections: 'Application ID' with a text input containing 'esp32-1ch-test-01'; 'Description' with a text input containing 'Test application for ESP32 1-CH gateway'; 'Application EUI' with a text input containing 'EUI issued by The Things Network'; and 'Handler registration' with a dropdown menu set to 'ttn-handler-us-west'. At the bottom right, there are 'Cancel' and 'Add application' buttons.

*Hard time seeing? Click on the image for a closer look!*

## Create and Configure a Device

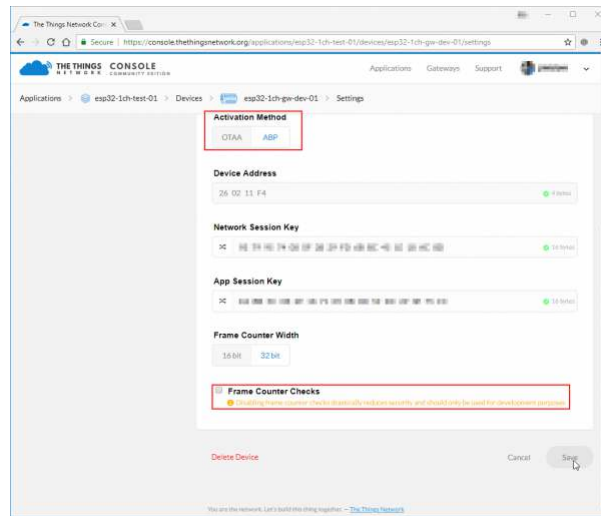
Next register a device in your application. Under the "**Devices**" section, click "**register device**".

This is again pretty simple. Fill out a unique device ID, click the "**generate**" button under "**Device EUI**" to automatically generate a EUI. Then click "**Register.**"

A screenshot of the The Things Network Console showing the 'Register Device' form. The form has four main sections: 'Device ID' with a text input containing 'esp32-1ch-gw-dev'; 'Device EUI' with a text input containing 'this field will be generated' and a 'generate' button; 'App Key' with a text input containing 'this field will be generated' and a 'generate' button; and 'App EUI' with a dropdown menu. At the bottom right, there are 'Cancel' and 'Register' buttons.

*Hard time seeing? Click on the image for a closer look!*

This example sketch only supports **ABP activation**, so you'll need to modify that in the device settings. In the "Device Overview" page, click "Settings". From there, under "Activation Method" click **ABP**. I also recommend **disabling Frame Counter Checks**. The gateway is capable of frame-counter checks, but it can get out of sync -- especially if you have another gateway nearby.



*Hard time seeing? Click on the image for a closer look!*

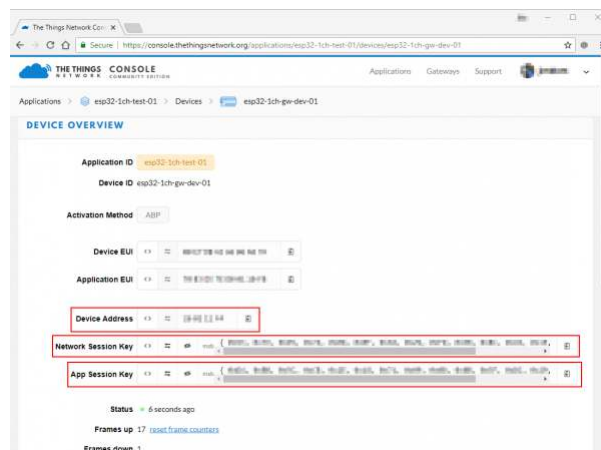
Save the settings and go back to your **"Device Overview"** page. You should see new keys, including **"Network Session Key"**, **"App Session Key"**, and **"Device Address."** These will need to be plugged into the device sketch and uploaded to the device - which we will cover in the next section.

## Update the Example Code

With your application, network session, and device keys in-hand, you're ready to finish configuring the ESP32 LoRa device sketch and start posting data!

On the **"Device Overview"** page, click the **"code"** symbol ( <> ) next to **"Network Session Key"** and **"App Session Key"** -- this will make the key visible and display it as a 16-byte array. Copy each of those keys, and paste them in place of the {PASTE KEY HERE} place-holders. **"Network Session Key"** should be pasted into the NWKSKY variable and **"App Session Key"** should be pasted into the APPSKY variable.

The DEVADDR variable expects a single 32-bit variable, so copy the **"Device Address"** key as shown and paste that into the PASTE\_DEV\_ADDR\_HERE placeholder.



*Hard time seeing? Click on the image for a closer look!*

Here's an example of what your three constants should look like once done:

```

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWKSKEY[16] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xBD, 0xEF, 0x01,
0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM APPSKEY[16] = { 0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10, 0xEE,
0xDC, 0xAA, 0x98, 0x76, 0x54, 0x32, 0x10 };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x01234567;

```

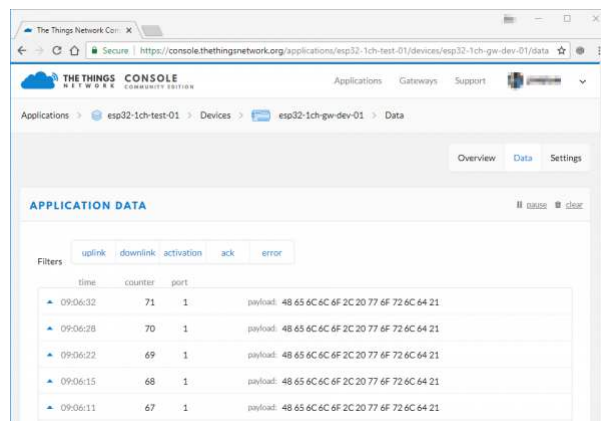
And that's it! Now upload the code to your LoRa Gateway 1-Channel board (the one that is supposed to be the device, that is!)

## Testing the Code

After setup, the device should immediately send a "Hello, World" message. It'll continue to send a message every minute or any time the "0" button on the board is pressed.

To check if your gateway is receiving the message, you can either check the Serial Monitor or monitor the ESP Gateway Config page served by the gateway. Every time a message is received it should be added to the "Message History" table.

If messages are getting through to your gateway, click the **"Data"** tab on your device to check for new messages.

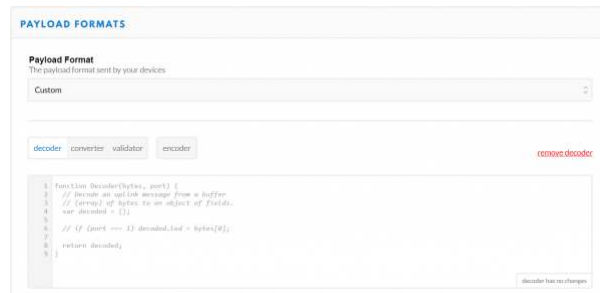


*Hard time seeing? Click on the image for a closer look!*

The message's payload -- which was encrypted between leaving the device and getting into your application -- should be a series of hex values corresponding to "Hello, world".

## Decoding the Message

You may have noticed in the *Application Data* window that your payload is shown in raw bytes. In order to see "Hello, World!" encoded in ASCII, the way you sent it, you'll need to decode the payload. The Things Network includes tools for doing this right in the console! Navigate to the *Application Overview* page for your application and click on the *Payload Formats* tab. This menu allows you to write functions which will be applied to all incoming packets for this application.



Hard time seeing? Click on the image for a closer look!

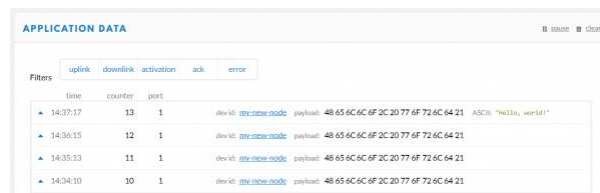
So let's write our own decoder. We need to take the raw byte data and return a string that contains all of the characters corresponding to each byte. Take a look at this solution and then we'll walk through it:

```
function Decoder(bytes, port) {
  return {
    ASCII: String.fromCharCode.apply(null, bytes)
  };
}
```

`Decoder` is a Javascript function that The Things Network has already set up for us. It takes two arguments called *bytes*, an array containing our payload, and *port*, the LoRaWAN™ "FPort" of the packet. FPort identifies the end application or service that the packet is intended for. Port 0 is reserved for MAC messages. We don't need to know anything about the port number for our example.

We can return any value that we want from the `Decoder` function and it will appear alongside our payload in the *Application Data* window. In the example above, I've created a new property called "ASCII" which is equal to `String.fromCharCode.apply(null, bytes)`. To break this down a little more, we're returning a new String object called "ASCII," and we're using the Javascript `apply()` method to call `fromCharCode()` with the argument `bytes` and stuff the result into our new String. The `fromCharCode()` method simply steps through each byte in the array `bytes` (which, remember, contains our payload) and returns the ASCII character represented by that character code.

After copying the above code into your decoder function, scroll down and click the *save payload functions* button. Now return to the *Application Data* window and you should see that all packets received after the decoder function was changed now have a new property:



Hard time seeing? Click on the image for a closer look!

Our packet has been decoded! Excellent!

## Troubleshooting

If messages are not getting to your gateway, first make sure the **channel and spread factor** match up. Left unchanged, the device example code should be sending data out on the 903.9MHz channel at a spread factor of 7 (That's assuming you've set the LMIC library to `CFG_us915`. If it's set to a European frequency spectrum, it'll be

868.1MHz, SF7).

You can use the gateway's web server to adjust these setting on the fly. Note that the channel numbers should sequentially match the `freqs` array in `loraModem.h` -- e.g. channel 0 should be 903.9MHz (again, assuming US frequencies).

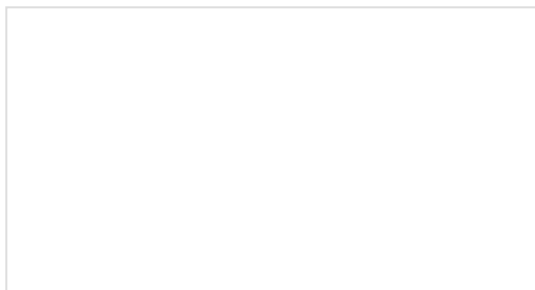
If messages are getting to your gateway, but not showing up on your TTN device console, consider changing the `_TTNSERVER` variable in "`ESP-sc-gway.h`". I haven't had success with the us-west router, but "`router.eu.thethings.network`" has worked perfectly (even in the US).

## Resources and Going Further

Thanks for coming on this single-channel LoRaWAN gateway journey with us! Here are a few links and documents that might prove handy as you continue exploring the world of LoRa with this board:

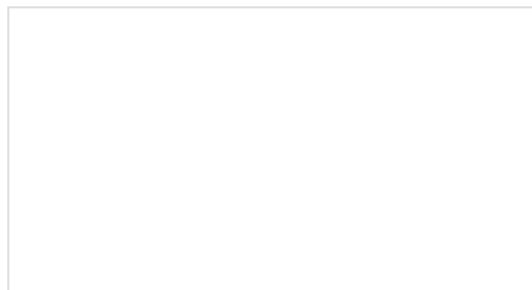
- The Things Network
- Product Repository -- GitHub repository where you can find all of our latest hardware and software design files.
- **Hardware**
  - Schematic -- PDF schematic
  - Eagle Files -- PCB design files
  - ESP32 WROOM Datasheet -- Datasheet for the ESP32 WROOM module
  - RFM95W Datasheet -- Datasheet for the RFM95W LoRa radio module
- **Software**
  - ESP-1ch-Gateway-v5.0 -- Single-Channel LoRaWAN Gateway Arduino firmware used in this tutorial.
  - Arduino-LMIC library -- LoRaMAC-in-C (LMIC) Arduino library used to create LoRaWAN devices.
    - ESP-1CH-TTN-Device-ABP -- Single-channel LoRaWAN device example using the Arduino-LMIC library.
- **Libraries Archived for this Tutorial**
  - LoRa Gateway 1-Channel Board Definition (ZIP)
  - ESP-1ch-Gateway-v5.0 (ZIP)
  - Arduino-LMIC from mcci-catena (ZIP)
  - Single-Channel LoRaWAN Device Example (ZIP)

Need Inspiration? Check out some of these fun IoT tutorials from SparkFun:



### Photon Battery Shield Hookup Guide

The Photon Battery Shield has everything your Photon needs to run off, charge, and monitor a LiPo battery. Read through this hookup guide to get started using it.



### Raspberry Pi 3 Starter Kit Hookup Guide

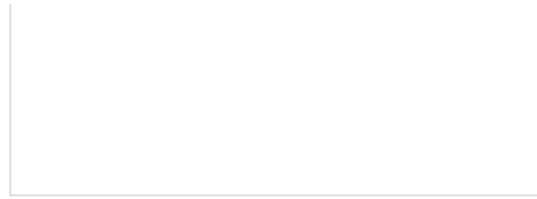
Guide for getting going with the Raspberry Pi 3 Model B and Raspberry Pi 3 Model B+ starter kit.





## ESP32 Environment Sensor Shield Hookup Guide

SparkFun's ESP32 Environment Sensor Shield provides sensors and hookups for monitoring environmental conditions. This tutorial will show you how to connect your sensor suite to the Internet and post weather data online.



## MicroMod nRF52840 Processor Hookup Guide

Get started with the MicroMod nRF52840 Processor following this guide.