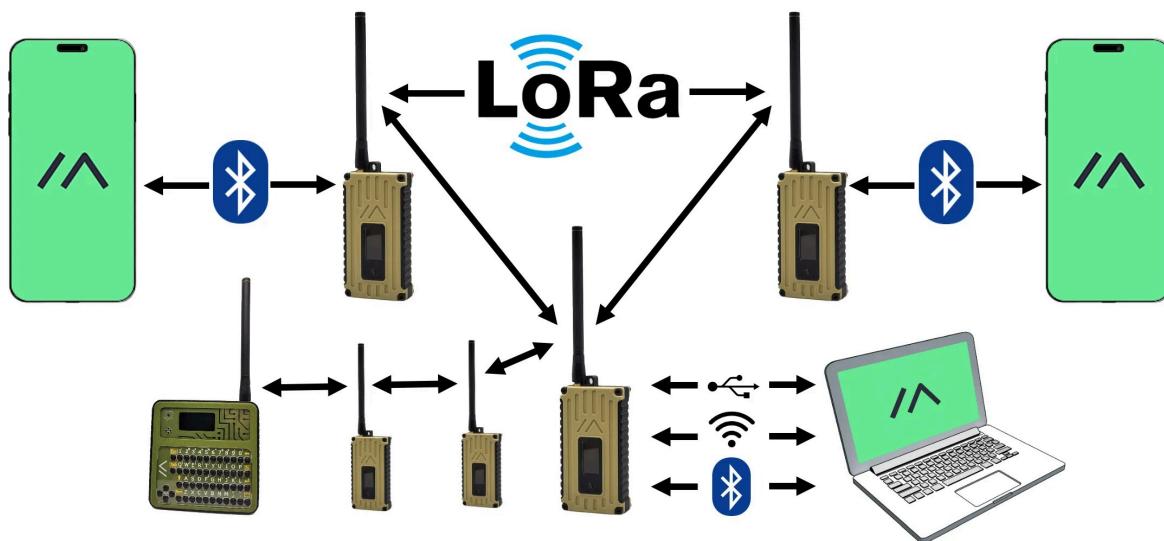


Introduction

Meshtastic® is a project that enables you to use inexpensive LoRa radios as a long range off-grid communication platform in areas without existing or reliable communications infrastructure. This project is 100% community driven and open source!



Features

- Long range (254km record by kboxlabs)
- No phone required for mesh communication
- Decentralized communication - no dedicated router required
- Encrypted communication
- Excellent battery life
- Send and receive text messages between members of the mesh
- Optional GPS based location features

And more!

How it works

Meshtastic utilizes LoRa, a long-range radio protocol, which is widely accessible in most regions without the need for additional licenses or certifications, unlike HAM radio operations.

These radios are designed to rebroadcast messages they receive, forming a mesh network. This setup ensures that every group member, including those at the furthest distance, can receive messages. Depending on the settings employed, the Meshtastic mesh network can support up to 100 devices concurrently.

Additionally, Meshtastic radios can be paired with a single phone, allowing friends and family to send messages directly to your specific radio. It's important to note that each device is capable of supporting a connection from only one user at a time."

If you are interested in a more technical overview of how Meshtastic works, visit the overview section below:

[Technical Overview](#)

Contributors

Meshtastic is an open source project available on GitHub. Our generous volunteers donate their personal time to write and maintain this codebase. If you would like to contribute see our GitHub, join our Discord server, and read up on our forum.

[Contribute!](#)

Start using Meshtastic

Hopefully your "Getting Started" experience is straight forward and headache free. If you encounter any issues, please consider updating our documentation to improve future user experiences or reach out on the forum or Discord.

Our support is 100% volunteer based. We are passionate about the project and hope to help newcomers become Meshtastic experts!

[Getting Started](#)

Overview

How it works

When you send a message on your Meshtastic companion app, it is relayed to the radio using Bluetooth, Wi-Fi/Ethernet or serial connection. That message is then broadcasted by the radio. If it hasn't received a confirmation from any other device after a certain timeout, it will retransmit the message up to three times.

When a receiving radio captures a packet, it checks to see if it has heard that message before. If it has it ignores the message. If it hasn't heard the message, it will rebroadcast it.

For each message a radio rebroadcasts, it marks the "hop limit" down by one. When a radio receives a packet with a hop limit of zero, it will not rebroadcast the message.

The radio will store a small amount of packets (around 30) in its memory for when it's not connected to a client app. If it's full, it will replace the oldest packets with newly incoming text messages only.

Radio Settings

ⓘ INFO

Meshtastic is **not** LoRaWAN, Helium or TTN (TheThingsNetwork). Meshtastic uses the full spectrum frequency range designated to LoRa technology per region. This allows for several hundred possible frequency channels in the US region alone.

ⓘ INFO

Power limits will generally be lifted in the software if `is_licensed` is set to `true`. See HAM Mode for more information.

Europe Frequency Bands

433 MHz

The maximum power allowed for Europe is +10 dBm ERP (Effective Radiated Power).

The band range is from 433 to 434 MHz.

There are four channels defined with the standard radio preset `LongFast`. After factory reset the radio will be set to channel 4

with a center frequency of 433.875 MHz.

868 MHz

The maximum power allowed for Europe is +27 dBm ERP (Effective Radiated Power).

The band range is from 869.40 to 869.65 MHz. This is less than the 863–870 MHz range defined as SRD Band, but allows for a higher ERP and a duty cycle of 10%.

There is one channel defined with the standard radio preset **LongFast**. After factory reset the radio will be set to channel 1 with a center frequency of 869.525 MHz.

North America Frequency Bands

915 MHz (ISM Band)

The maximum output power for North America is +30 dBm ERP (Effective Radiated Power).

The band range is from 902 to 928 MHz.

There are 104 channels defined with the standard radio preset **LongFast**. After factory reset the radio will be set to channel 20 with a center frequency of 906.875 MHz.

Data Rates

Considerations

Various data-rate options are available when configuring a channel and are inversely proportional to the theoretical range of the devices.

Spreading Factor (SF) - How much we "spread" our data over time.

Each step up in Spreading Factor doubles the airtime to transmit.

Each step up in Spreading Factor adds about 2.5db extra link budget.

Bandwidth - How big of a slice of the spectrum we use.

Each doubling of the bandwidth is almost 3db less link budget.

Bandwidths less than 31 may be unstable unless you have a high quality Crystal Oscillator.

Coding Rate - How much redundancy we encode to resist noise.

Increasing coding rate increases reliability while decreasing data-rate.

4/5 - 1.25x overhead

4/6 - 1.5x overhead

4/7 - 1.75x overhead

4/8 - 2x overhead

Presets

We have eight LoRa radio presets. These are the most common settings and have been proven to work well:

Channel setting	Alt Channel Name	Data-Rate	SF / Symbols	Coding Rate	Bandwidth	Link Budget
Short Range / Fast	Short Fast	10.94 kbps	7 / 128	4/5	250	143dB
Short Range / Slow	Short Slow	6.25 kbps	8 / 256	4/5	250	145.5dB
Medium Range / Fast	Medium Fast	3.52 kbps	9 / 512	4/5	250	148dB
Medium Range / Slow	Medium Slow	1.95 kbps	10 / 1024	4/5	250	150.5dB
Long	Long Fast	1.07	11 /	4/5	250	153dB

Channel setting	Alt Channel Name	Data-Rate	SF / Symbols	Coding Rate	Bandwidth	Link Budget
Range / Fast		kbps	2048			
Long Range / Moderate	Long Moderate	0.34 kbps	11 / 2048	4/8	125	156dB
Long Range / Slow	Long Slow	0.18 kbps	12 / 4096	4/8	125	158.5dB
Very Long Range / Slow	Very Long Slow	0.09 kbps	12 / 4096	4/8	62.5	161.5dB

ⓘ NOTE

The link budget used by these calculations assumes a transmit power of 22dBm and an antenna with 0dB gain. Adjust your link budget assumptions based on your actual devices. Data-rate in this table is the theoretical max but doesn't account for

packet headers, hops and re-transmissions. Calculations based on data from the official Semtech LoRa calculator.

Custom Settings

Custom settings can be applied by using supported software.

After applying the settings, you will need to restart the device. After your device is restarted, it will generate a new crypto key and you will need to share the newly generated QR Code or URL to all your other devices.

Some example settings:

Data-rate	SF / Symbols	Coding Rate	Bandwidth	Link Budget	Note
37.50 kbps	6 / 64	4/5	500	129dB	Fastest possible speed
3.125 kbps	8 / 256	4/5	125	143dB	
1.953 kbps	8 / 256	4/8	125	143dB	

Data-rate	SF / Symbols	Coding Rate	Bandwidth	Link Budget	Note
1.343 kbps	11 / 2048	4/8	500	145dB	
1.099 kbps	9 / 512	4/8	125	146dB	
0.814 kbps	10 / 1024	4/6	125	149dB	
0.610 kbps	10 / 1024	4/8	125	149dB	
0.488 kbps	11 / 2048	4/6	125	152dB	
0.073 kbps	12 / 4096	4/5	31	160dB	Twice the range and/or coverage of "Long Slow", low resilience

Data-rate	SF / Symbols	Coding Rate	Bandwidth	Link Budget	Note
					to noise
0.046 kbps	12 / 4096	4/8	31	160dB	Twice the range and/or coverage of "Long Slow", high resilience to noise

The link budget used by these calculations assumes a transmit power of 17dBm and an antenna with 0dB gain. Adjust your link budget assumptions based on your actual devices.

These channel settings may not have been tested. Use at your own discretion. Share on <https://meshtastic.discourse.group> with your successes or failure.

Cryptography

The pre-shared key (PSK) used by the devices can be an AES128 or AES256 sequence. Alternatively, encryption can be turned off, which may be useful if you are operating under a Ham Radio

license.

Mesh Broadcast Algorithm

Current Algorithm

The routing protocol for Meshtastic is really quite simple (and sub-optimal). If you want to test its theoretical performance, you can have a look at the simulator. The protocol is heavily influenced by the mesh routing algorithm used in RadioHead (which was used in very early versions of this project). It has four conceptual layers.

A Note About Protocol Buffers

Because we want our devices to work across various vendors and implementations, we use Protocol Buffers pervasively. For purposes of this document you mostly only need to consider the MeshPacket and Sub-packet message types.

Layer 0: LoRa Radio

All data is converted into LoRa symbols which are sent to the radio for transmission. The details are described elsewhere, but it is worth noting that in addition to the converted packet bytes described below, there is also a preamble sent at the start of any data packet.

This preamble allows receiving radios to synchronize clocks and start framing. We use a preamble length of 16, which is longer than the minimum preamble length of 8, to let SX126x LoRa receivers sleep for a while, which lowers power consumption.

After the preamble comes the LoRa Physical Header, which contains information about the packet length as well as a sync word to distinguish networks. For Meshtastic, it is set to `0x2B`.

Layer 1: Unreliable Zero Hop Messaging

This layer is conventional non-reliable LoRa packet transmission. A

packet generated by a Meshtastic device has the following representation before encoding for transmission:

Offset	Length	Type	Usage
0x00	4 bytes	Integer	Packet Header: Destination. The destination's unique NodeID. <code>0xFFFFFFFF</code> for broadcast.
0x04	4 bytes	Integer	Packet Header: Sender. The sender's unique NodeID.
0x08	4 bytes	Integer	Packet Header: The sending node's unique packet ID for this packet.
0x0C	1 byte	Bits	Packet Header: Flags. See the header flags for usage.
0x0D	1 byte	Bits	Packet Header: Channel hash. Used as hint for decryption for the receiver.
0x0E	2 bytes	Bytes	Packet Header: Padding for memory alignment.

Offset	Length	Type	Usage
0x10	Max. 237 bytes (excl. protobuf overhead)	Bytes	Actual packet data. Unused bytes are not transmitted.

Packet Header Flags

Index	# of Bits	Usage
0	3	HopLimit (see note in Layer 3)
3	1	WantAck
4	1	ViaMQTT (packet came via MQTT)
5 .. 7	3	Currently unused

Usage Details

Packet Header: is described directly by the `PacketHeader` class in the C++ source code. But indirectly it matches the first portion of the `MeshPacket` protobuf definition. Note that the packet header is not encoded using a protobuf, but is sent as raw bytes. This both saves airtime and allows receiving radio hardware to

optionally filter packets before waking the main CPU.

Packet Header - NodeIDs: are constructed from the bottom four bytes of the MAC address of the Bluetooth address. Because the OUI is assigned by the IEEE, and we currently only support a few CPU manufacturers, the upper byte is de-facto guaranteed unique for each vendor. The bottom 3 bytes are guaranteed unique by that vendor.

Packet Header - Unique ID: The ID is a large, 32 bit ID to ensure there is enough unique state to protect an encrypted payload from attack.

Payload: An encrypted and packed protobuf encoding of the SubPacket protobuf. Only the SubPacket is encrypted, while headers are not. This allows the option of eventually allowing nodes to route packets without knowing anything about the encrypted payload. For more information, see the encryption and Protobuf API Reference. Any data past the maximum length is truncated.

Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA)

Meshtastic adopts CSMA/CA, similar as to what is used in WiFi. This means that all transmitters must perform Channel Activity Detection (CAD) before attempting to transmit. If the channel is considered busy, the node will wait until it is not anymore. Since once the channel becomes idle multiple nodes might want to start

transmitting, a node has to wait a random multiple of slot times. The slot time is the time needed to reliably perform CAD. The amount of slot times to wait is randomly picked from a contention window (CW), which size depends on the current channel utilization. The contention window is larger for a higher channel utilization, in order to limit the chance of collisions.

Layer 2: Reliable Zero Hop Messaging

This layer adds reliable messaging between the node and its immediate neighbors only.

The default messaging provided by Layer 1 is extended by setting the `WantAck` flag in the MeshPacket protobuf. If `WantAck` is set, the following documentation from mesh.proto applies:

This packet is being sent as a reliable message, we would prefer it to arrive at the destination. We would like to receive an ACK packet in response.

Broadcast messages treat this flag specially: Since ACKs for broadcasts would rapidly flood the channel, the normal ACK behavior is suppressed. Instead, the original sender listens to see if at least one node is rebroadcasting this packet (because naive flooding algorithm). If it hears that, the odds (given typical LoRa topology) are very high that every node should eventually receive the message. So FloodingRouter.cpp generates an implicit ACK which is delivered to the original sender. If after some time we don't hear anyone rebroadcast

our packet, we will timeout and re-transmit, using the regular resend logic.

If a transmitting node does not receive an ACK (or NAK) packet after a certain expiration time, it will use Layer 1 to attempt a re-transmission of the sent packet. A reliable packet (at this 'zero hop' level) will be resent a maximum of three times. If no ACK or NAK has been received by then the local node will internally generate a NAK (either for local consumption or use by higher layers of the protocol). The re-transmission expiration time is based on the maximum time it would take to receive an (implicit) ACK, taking the airtime of the sent packet and any processing delay into account. For direct messages, the intended recipient will also send a real ACK all the way back to the original sender, but the device will only retransmit when it received no ACK at all.

Layer 3: (Naive) Flooding for Multi-Hop Messaging

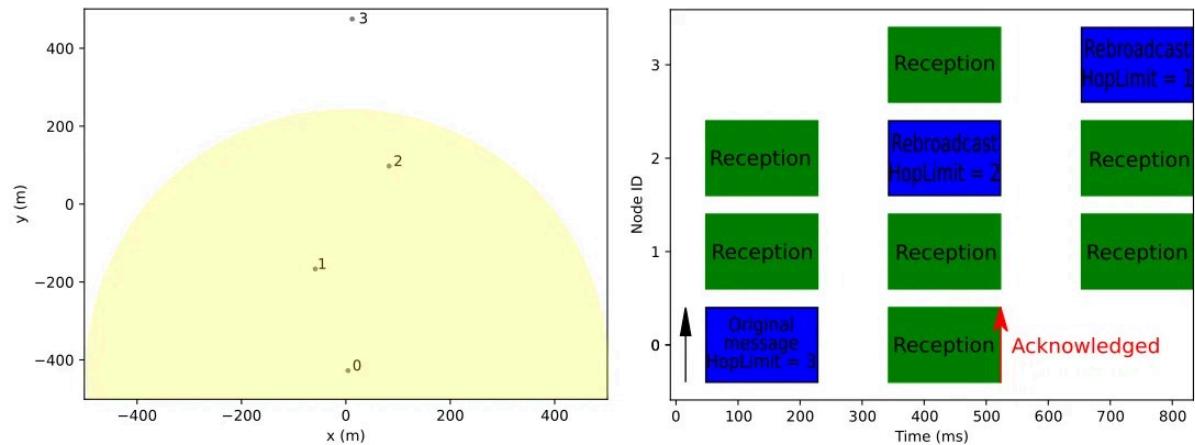
Given our use-case for the initial release, most of our protocol is built around flooding. The implementation is currently 'naive' and doesn't try to optimize flooding, except by abandoning re-transmission once a node has seen a nearby receiver ACK the packet it's trying to flood. This means that up to N re-transmissions of a packet could occur in an N node mesh.

If any mesh node sees a packet with a HopLimit other than zero, it will decrement that HopLimit and attempt re-transmission on

behalf of the original sending node. In order to promote letting nodes that are further away flood the message, such that the message eventually reaches farther, the contention window (see Layer 1) for a flooding message depends on the Signal-to-Noise Ratio (SNR) of the received packet. The CW size is small for a low SNR, such that nodes that are further away are more likely to flood first and closer nodes that hear this will refrain from flooding.

Example

Below you see an example topology consisting of four nodes, where at a certain point node 0 wants to send a broadcast message. Due to limited coverage, it only reaches nodes 1 and 2. Since node 2 is farther away, its SNR is lower and therefore starts rebroadcasting earlier than 1. After node 0 received this rebroadcast, its message is acknowledged. Note that a message is already acknowledged once a rebroadcast from any Meshtastic node (whether or not it has the same encryption key) is received. Since node 1 heard the rebroadcast by 2, it will not rebroadcast again. Node 3 heard the message for the first time and the HopLimit is not yet zero, so it starts a rebroadcast for potential other receivers.



Meshtastic Encryption

Cryptography is tricky, so we've tried to 'simply' apply standard crypto solutions to our implementation. However, the project developers are not cryptography experts.

Based on comments from reviewers (see below), here are some tips for usage of these radios, so that you may know the level of protection offered:

It is pretty likely that the AES256 security is implemented 'correctly' and an observer will not be able to decode your messages.

Warning: If an attacker is able to get one of the radios in their possession, they could either a) extract the channel key from that device or b) use that radio to listen to new communications.

Warning: If an attacker is able to get the "Channel QR code/URL" that you share with others - that attacker could then be able to read any messages sent on the channel (either tomorrow or in the past - if they kept a raw copy of those broadcast packets)

The current implementation provides optional confidentiality to members of a configured network:

Encryption is implemented in devices/nodes with network-wide encryption keys.

Encryption is optional and is turned off when devices are in 'Ham

mode'.

There is no encryption supported in the clients (iOS, Android) to facilitate distribution as mass market software.

Pairing from client-to-device is by:

direct USB cable

BT pairing

Devices are 'promiscuous' and will pair with any near-by client.

Network confidentiality requires physical protection of all nodes.

Always keep in mind xkcd's note on encryption.

If you are a cryptography expert, please review these notes and our questions below. Can you help us by reviewing our notes below and offering advice? We will happily give as much or as little credit as you wish ;-).

Consider our existing solution 'alpha' and probably fairly secure against a not particularly aggressive adversary (but we can't yet make a more confident statement).

Notes for reviewers

If you are reviewing our implementation, this is a brief statement of our method.

We do all crypto at the SubPacket (payload) level only, so that all Meshtastic nodes will route for others - even those channels which are encrypted with a different key.

Mostly based on reading Wikipedia and using the modes the ESP32 provides support for in hardware.

We use AES256-CTR as a stream cypher (with zero padding on the last BLOCK) because it is well supported with hardware acceleration.

Our AES key is 128 or 256 bits, shared as part of the 'Channel' specification.

The node number concatenated with the packet number is used as the NONCE. This nonce will be stored in flash in the device and should essentially never repeat. If the user makes a new 'Channel' (i.e. picking a new random 256 bit key), the packet number will start at zero.

The packet number is sent in cleartext with each packet. The node number can be derived from the "from" field of each packet. (Cleartext is acceptable because it merely provides IV for each encryption run)

Each 16 byte BLOCK for a packet has an incrementing COUNTER. COUNTER starts at zero for the first block of each packet.

The IV for each block is constructed by concatenating the NONCE as the upper 96 bits of the IV and the COUNTER as the bottom 32 bits. Since our packets are small counter portion will really never be higher than 32 (five bits).

Comments from reviewer #1

This reviewer is a cryptography professional, but would like to

remain anonymous. We thank them for their comments ;-):

I'm assuming that meshtastic is being used to hike in places where someone capable is trying to break it - like you were going to walk around DefCon using these. I spent about an hour reviewing the encryption, and have the following notes:

The write-up isn't quite as clear as the code.

The code is using AES-CTR mode correctly to ensure confidentiality.

The comment for initNonce really covers the necessary information.

I think the bigger encryption question is "what does the encryption need to do"? As it stands, an attacker who has yet to capture any of the devices cannot reasonably capture text or location data. An attacker who captures any device in the channel/mesh can read everything going to that device, everything stored on that device, and any other communication within the channel that they captured in encrypted form. If that capability basically matches your expectations, it is suitable for whatever adventures this was intended for, then, based on information publicly available or widely disclosed, the encryption is good. If those properties are distressing (like, device history is deliberately limited and you don't want a device captured today to endanger the information sent over the channel yesterday) we could talk about ways to achieve that (most likely synchronizing time and replacing the key with its own SHA256 every X hours, and ensuring the old key is not

retained unnecessarily).

Two other things to keep in mind are that AES-CTR does not itself provide authenticity (e.g. an attacker can flip bits in replaying data and scramble the resulting plaintext), and that the current scheme gives some hints about transmission in the size. So, if you worry about an adversary deliberately messing-up messages or knowing the length of a text message, it looks like those might be possible.

I'm guessing that the network behaves somewhat like a store-and-forward network - or, at least, that the goal is to avoid establishing a two-way connection to transmit data. I'm afraid I haven't worked with mesh networks much, but remember studying them briefly in school about ten years ago.

Range Tests

Ground

Current Ground Record: 254km

Range: 254km (158 miles)

Record Holders: *kboxlabs*

Source: Meshtastic Discourse

Modem Settings

Default Long_Fast

Frequency: 915MHz

Bandwidth: 250

Spread Factor: 11

Coding Rate: 4/8

Node A

Device: RAK4631 Core with RAK 5005-O Base Board

Firmware Version: 2.1.17

Antenna: 902-928MHz 5.8 dBi Slinkdsco Outdoor

Node B

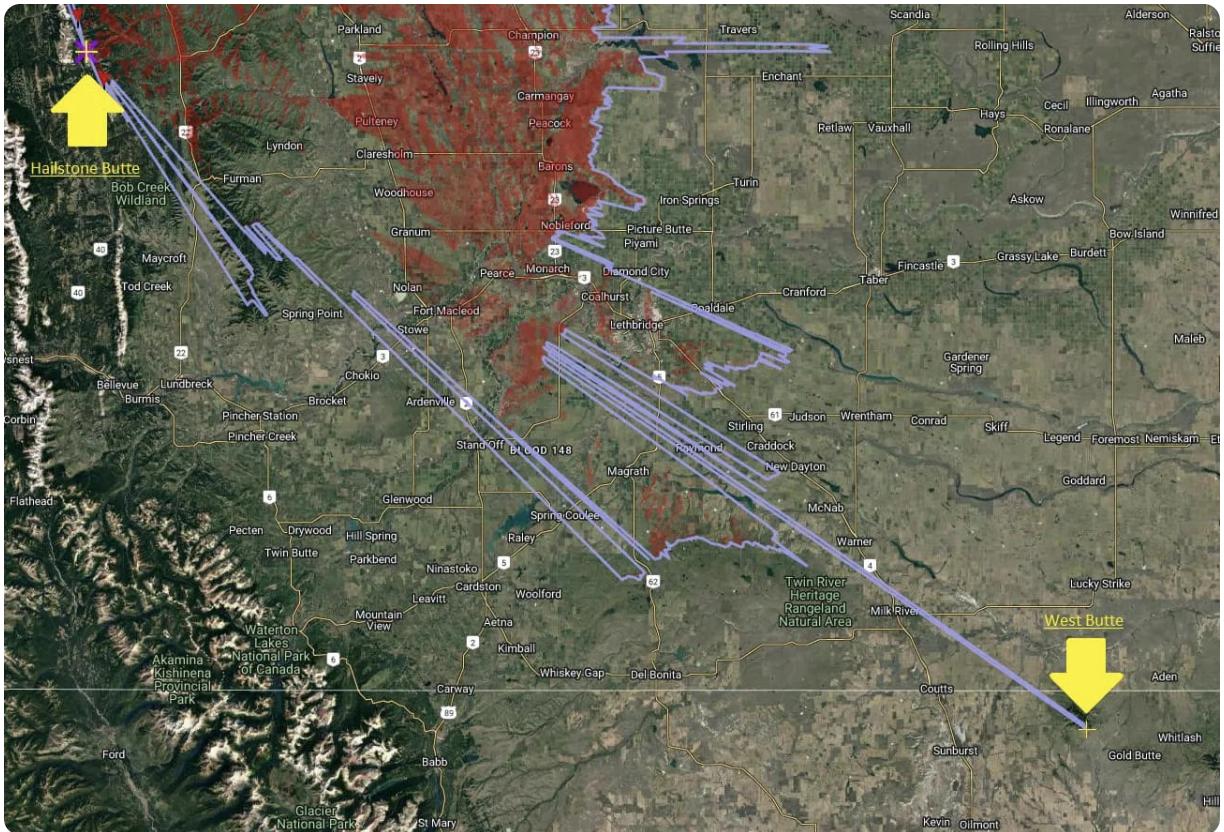
Device: RAK4631 Core with RAK 19003 Mini Base Board

Firmware Version: 2.1.18

Antenna: Standard LoRa 915MHz 60mm 2dBi Omnidirectional







Previous Record 166km

Range: 166km (103 miles)

Record Holder: *PuzzledPancake*

Source: Meshtastic Discourse

Modem Settings

Frequency: 868MHz

Bandwidth: 125

Spread Factor: 12

Coding Rate: 4/8

Node A

Device: LILYGO TTGO T-Beam w/ SX1262

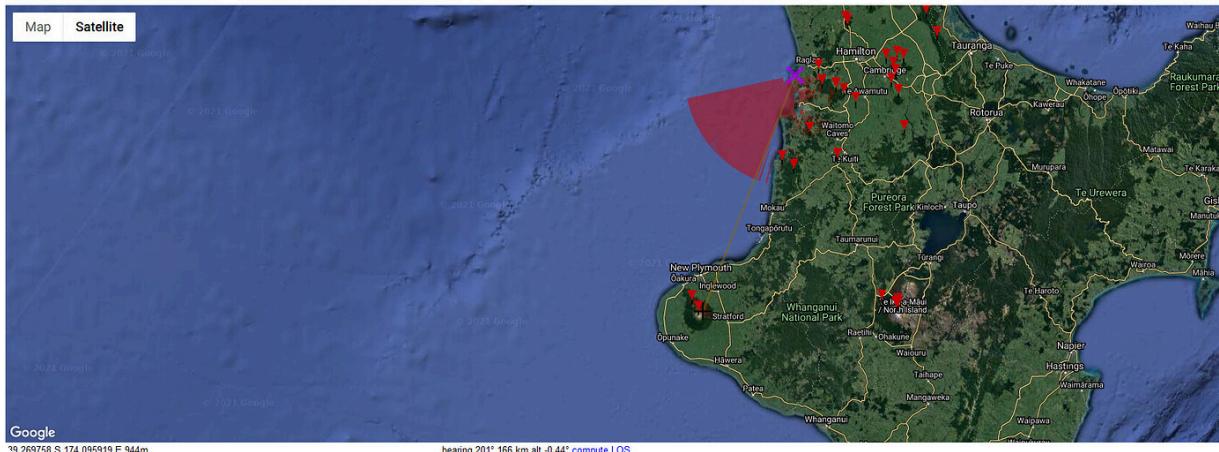
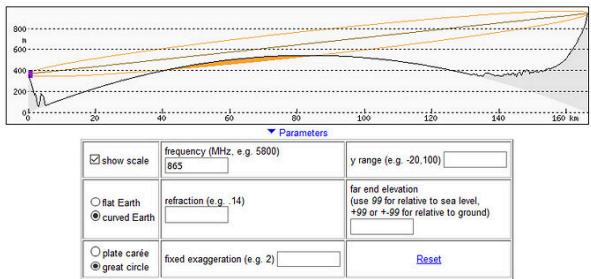
Firmware Version: 1.2

Antenna: 868MHz 5dBi Antenna

Node B

Device: LILYGO TTGO T-Beam w/ SX1262

Firmware Version: 1.2
Antenna: 868MHz Vertical 6dBi







Resources Used

<http://www.heywhatsthat.com>

Air

Current Air Record: 206km

Range: 206km (128 miles)

Record Holders: *StarWatcher, CVR, rook, kboxlabs*

Source: Meshtastic Discourse

Modem Settings

Default Long_Fast

Frequency: 915MHz

Bandwidth: 250

Spread Factor: 11

Coding Rate: 4/8

Node A

Device: LILYGO TTGO T-Beam

Firmware Version: 2.1.10

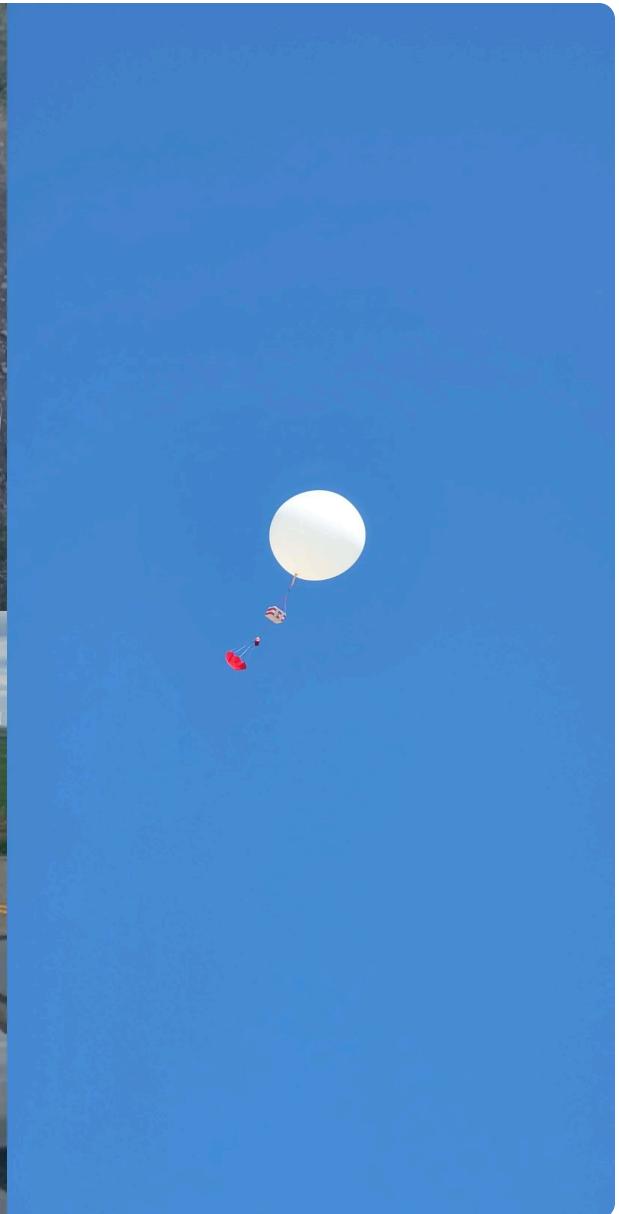
Antenna: Stock Antenna

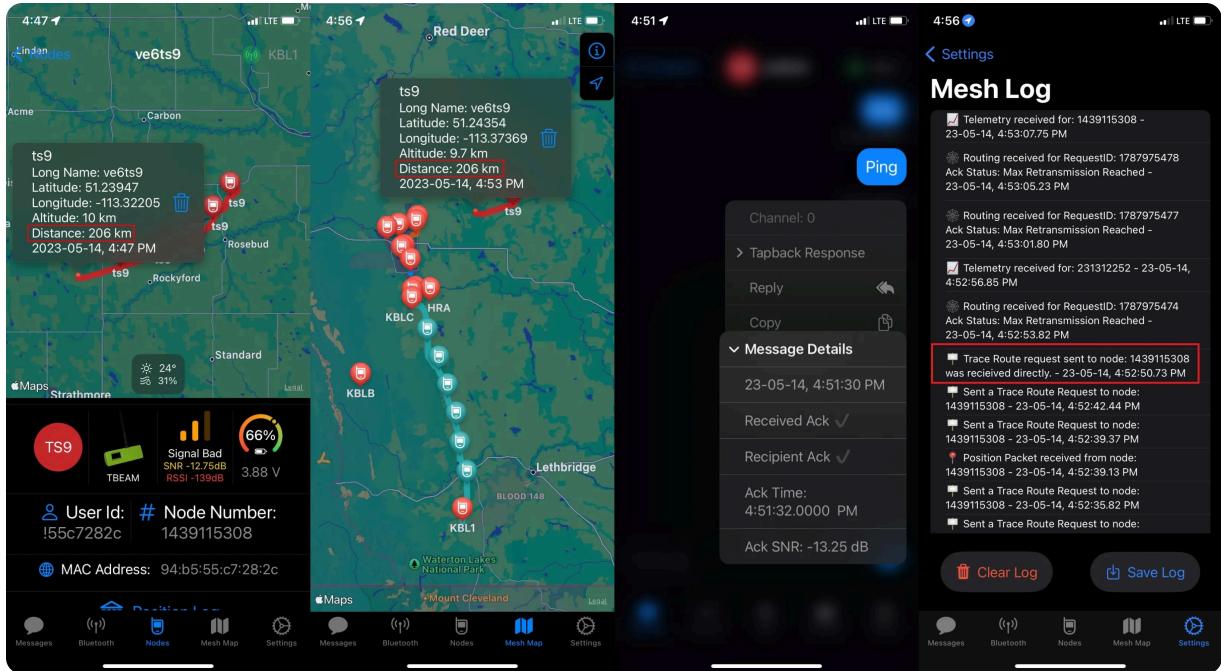
Node B

Device: LILYGO TTGO T-Beam

Firmware Version: 2.1.10 (modified to place GPS in flight mode)

Antenna: Stock Antenna





Contributing to Meshtastic

Volunteer Based Development

Meshtastic is a team of volunteers, and as such there are always plenty of ways to help. This project gets great contributions from people in their off hours. Those contributors work on the features they are interested in. It is a very open and welcoming developer community, and we are always looking for help to improve Meshtastic.

If you're a developer, there's plenty stuff to do. Dig in!

If you're interacting with Meshtastic radios, we could use help with testing, documenting, and providing feedback.

If you're into Web Development, check out the different web repos.

If you're into Kotlin and Android, check out the link to the repo below.

If you're into Python, check out the link to the repo below

If you're into SwiftUI, check out the link to the repo below

If you're into Ham Radio and LoRa, then this is a great project for you!

... basically... we would love to have your help and feedback

There are several developers, testers, and active users on Discord.

Meshtastic Ecosystem

The Meshtastic ecosystem leverages a variety of technologies and repositories to provide a robust decentralized communication platform. This ecosystem is continuously evolving, thanks to the collaborative effort of developers, contributors, and the community, aiming to enhance and expand Meshtastic's capabilities for decentralized off-grid communication.

Key components include:

Protocol Buffers: The backbone of communication and interactions, with changes and definitions managed in the Meshtastic Protobuf Definitions repository. For detailed information, see the Protobuf API Reference.

Device Firmware: Development for ESP32, nRF52, and RP2040 based devices occurs in the firmware repository, focusing on C and C++ code for hardware interaction and communication.

Firmware Modules: Extend the core functionalities of devices and mesh networks, implemented mainly within the firmware repository. Modules are essential for adding new features and integrating devices within the ecosystem.

CLI Apps (Device Interface): The Meshtastic Python CLI enables interaction with device settings and functionalities, serving both as a utility and a library for application development.

Web and JavaScript Apps: Development for the hosted web

server on ESP32 devices is done in Meshtastic Web, with a JavaScript library for device interfaces available in Meshtastic JS.

Mobile and Desktop Apps: Android, iOS, iPadOS, and macOS applications provide user interfaces for interacting with Meshtastic devices. Development for Android is detailed in the Meshtastic-Android repository, while Apple platforms are supported by the Meshtastic-Apple repository.

Documentation: The source for the Meshtastic website and documentation resides in the meshtastic/meshtastic repository. For guidelines on maintaining documentation, visit the Maintaining Documentation page.

This ecosystem is continuously evolving, thanks to the collaborative effort of developers, contributors and the community, aiming to enhance and expand Meshtastic's capabilities for decentralized off-grid communication.

Supporting and Contributing to Meshtastic

The Meshtastic project benefits greatly from the contributions of our volunteers and the financial support from our community. As the project has evolved, the expansion of features and services has led to increased financial needs, including hardware for development, GitHub fees, hosting for public MQTT services, among others. This growth into a robust platform for decentralized

off-grid mesh networking solutions showcases the dedication of our developers and the generosity of our financial supporters.

How You Can Support Meshtastic Financially

If direct development contribution is not feasible for you, financial support is another invaluable way to help Meshtastic grow and thrive. We offer two primary options for monetary donations:

[Support on Open Collective](#)

[Sponsor on GitHub](#)

Fiscal Sponsorships

We're deeply thankful for the backing from the Open Collective, Open Source Collective, DigitalOcean, and Vercel, which has been instrumental in our project's sustainability and growth:

Open Collective & Open Source Collective: Our collaboration with these organizations offers a robust fiscal management framework and banking solutions, supporting our project alongside over three thousand other open source initiatives. Open Collective's transparent framework lets everyone see our finances,

including income, expenditures, and contributions on Meshtastic's Open Collective page. This transparency and support affirm that we're in excellent company.

DigitalOcean: Champions of open source, DigitalOcean supports Meshtastic through credits, which significantly aid our development, infrastructure, and testing efforts. This contribution is part of their commitment to fostering innovation and collaboration within the open source ecosystem.

Vercel: By covering our hosting costs, Vercel directly contributes to Meshtastic's growth, enhancing our web scalability and presence. Their sponsorship is crucial for our continued development and focus on creating a robust platform for decentralized communication.

FAQs

Overview

- › Where can I get additional help, ask questions, or bond with the Meshtastic community?

This site (which has a great search function) is the preferred place for up-to-date documentation. Many of our users and developers hang out on the Meshtastic Discord server where you may connect with like-minded people.

- › How can I contribute to Meshtastic?

Everyone contributes in a different way. Join the Meshtastic Discord and introduce yourself. We're all very friendly. If you'd like to pitch in some code, check out the Development menu on the left. If you'd like to contribute financially, please visit our page on Open Collective or you may choose to sponsor a developer. Check out Contributing for more details.

Firmware

- › Can I update my node's firmware over the mesh?

No, Meshtastic does not support OTA updates over LoRa.
Please visit Flash Firmware for update options.

Android Client

- › What versions of Android does the Meshtastic Android App require?

Minimum requirement is Android 5 (Lollipop 2014, first BLE support), however at least Android 6 (Marshmallow 2015) is recommended as Bluetooth is more stable. While Android 5/6 are officially supported by Meshtastic, it is *not* recommended that you purchase devices with these versions due to their limited OS support and limited battery life due to age. Many newer models exist that are very affordable. A good resource to use when researching affordable devices is the LineageOS Supported Devices List.

- › What does the icon next to the message mean?

Cloud with an up arrow - Queued on the app to be sent to your device.

Cloud only - Queued on the device to be sent over the mesh.

Cloud with a check mark - At least one other node on the mesh acknowledged the message.

Person with a check mark - The intended recipient of your direct message acknowledged the message.

Cloud crossed out - Not acknowledged or message error.

› How can I clear the message history?

Long press any message to select and show the menu with "delete" and "select all" buttons.

› After a fresh firmware install, my node is not connecting via Bluetooth. What should I do?

Try forgetting the Bluetooth connection from the Android Bluetooth Settings menu. Re-pair and try again. This is a security measure and there is no workaround for it. It prevents apps and other accessories from spoofing an existing accessory by un-pairing and "re-pairing" themselves without the users' knowledge.

Apple Clients

› What version of iOS/iPadOS/macOS does the Meshtastic App Require?

The Meshtastic App on Apple Clients require the following

minimum OS versions: iOS 16.2, iPadOS 16.2, and macOS 13.

› How do I get the Apple Meshtastic App?

See Apple Apps

› After a fresh firmware install, my node is not connecting via Bluetooth. What should I do?

Try forgetting the Bluetooth connection from the iOS/iPadOS/macOS System Settings. Re-pair and try again. This is a security measure and there is no workaround for it. It prevents apps and other accessories from spoofing an existing accessory by un-pairing and "re-pairing" themselves without the users' knowledge.

› Do the Apple applications provide an indication if the message was acknowledged on the mesh?

Yes, when the message is sent you will first see a Waiting to be acknowledged... status beneath the message. If the message is acknowledged by a node on the mesh this will update to Acknowledged in orange, which turns into grey when sending a direct message and the intended recipient acknowledged it. If no nodes have responded it will indicate Max Retransmission Reached. If there is an error, the status

will update to the appropriate error. Additionally, you can long press on the message and select `Message Details` to view the date/time sent, if ack was received and the time ack was received or the error (if there was one).

Channels

- › How do I share my Meshtastic Channel with other people?

Your Meshtastic client (Android, Apple, Web, or Python) will provide you a URL or QR code. You can email, text or print this URL or QR code and share it with people you want to join your Meshtastic Channel.

Python CLI

- › How do I find out more about installing (and using) Meshtastic via command line?

See our guide [here](#).

- › How do I find out more about using python to interact?

See our guide [here](#).

› What if I'm still having issues on Windows 10?

It's been reported that `App execution aliases` might conflict with one another and prevent python3 from being able to run properly. There is an example of a fix located [here](#).

Devices

› How do I turn off an ESP32 T-Beam based device?

Hold down the left PWR button for about 10 seconds and the display should turn off.

› How do I turn on an ESP32 T-Beam based device?

Push the left PWR button for about 1 second.

› Functionality of the T-Beam Buttons

T-Beam Buttons explained [here](#)

› Where do I purchase the device hardware?

Each supported device has a "Purchase Link".

› I have my hardware. How do I install the firmware and any

required drivers?

See our guide here.

› How do I update the firmware to the latest version?

Updating firmware varies with hardware. See [Flashing Firmware](#).

› My device has gone to sleep. Are received messages lost?

The LoRa radio on the node is still active and will wake up the CPU when the device is sleeping. If your phone is in range, the node will relay any messages your phone may have missed. If you're in range and your device is active, messages have not been lost.

› My device has gone to sleep and I can't send any messages.

Once the node wakes up from sleep, your phone will relay any delayed messages through your node and to the mesh network. Give it a few minutes and it'll do the right thing.

› How can I tell the device not to sleep?

See [Device Power Configuration](#) options.

- › I am in Europe and my device seems to stop transmitting after a while, what is going on?

Europe has an hourly duty cycle limit of 10% in the frequency band that Meshtastic uses. It might be that you hit this limit if you are sending a lot. You can confirm this by checking whether you see duty cycle limit errors in the serial log, Mesh Log (Apple apps) or Debug Panel (Android). To limit traffic, you can consider setting the device metrics and position update intervals higher. Alternatively, the device can be configured to override the duty cycle limit, but then you will violate the regulations.

- › Why does only one RAK Meshtastic Starter kit show up in my node list?

There was a bug where Meshtastic Starter kits were sent out with the same MAC address. With a single MAC address the devices all report as being the same device. Without the battery connected, flash the starter kit device(s) to any firmware > 1.2.59 and then do a factory reset, disconnect and reconnect the board and run `meshtastic --info`.

Amateur Radio (ham)

Meshtastic can be used by both unlicensed people and licensed HAM operators.

› What is the benefit of using a ham license with Meshtastic?

If you use your ham radio license with Meshtastic, consider both the privileges and restrictions:

Privileges

Increased Transmit Power

Up to 10W transmit power in the United States! 47 CFR 97.313(j)

Higher Gain Antennas

Restrictions

Plain-Text Only

On amateur radio bands, encryption is illegal. FCC Part 97.113.A.4

Lack of Privacy

As a ham operator, it is a requirement that you identify yourself by your call sign periodically when transmitting. Your call sign will be publicly transmitted at least once every 10 minutes at minimum. FCC Part 97.119.A

› How do I set my ham call sign?

- On Android navigate to Radio configuration -> User and set Long Name
- On iPhone navigate to Settings -> User and set Long Name
- Instructions for Enabling ham License from the Python CLI

Mesh

› Does Meshtastic use LoRaWAN?

No, Meshtastic uses LoRa peer to peer (p2p), which allows much more flexibility in how LoRa is utilized compared to LoraWan. Our messaging and position updates are far too "random" compared to LoRaWAN requirements.

› Will Meshtastic work with (insert LoRa service)

Meshtastic uses LoRa peer to peer (p2p), which has allowed us to customize how we use the protocol. Likely it will not work with the service you have in mind, but it may be possible to build a bridge between services using MQTT. That will require further development outside the scope of this project to implement.

› Can I locate a device via triangulation?

There is a community project that has worked out how to accomplish this.

Modules

› What are Modules?

Modules are features that expand the basic device functionality and/or integrate with other services.

› What modules do we have available?

A list of available modules is available here.

› I'd like to write a module. How do I get started?

API documentation for creating modules is available here.

Getting Started

Identify Hardware

 NOTE

This guide assumes that you have already purchased the devices you will be using with Meshtastic. If you haven't, you can check out our list of supported hardware to see your options.

Before you begin, it's important to determine which kind of hardware you're using. Meshtastic works with devices that have these types of Micro-Controller Units (MCU):

ESP32

The ESP32 chip is older and consumes more power than the nRF52 chip, but is equipped with both WiFi and Bluetooth. Supported ESP32 devices include:

LILYGO® TTGO T-Beam (>V1.1 recommended)

LILYGO® TTGO Lora (>V2.1 recommended)

Nano G1

Station G1

Heltec V3 and Wireless Stick Lite V3

RAK11200 Core module for RAK WisBlock modular boards

nRF52

The nRF52 chip is much more power efficient than the ESP32 chip and easier to update, but is only equipped with Bluetooth.

Supported nRF52 devices include:

RAK4631 Core module for RAK WisBlock modular boards

LILYGO® TTGO T-Echo

RP2040

The RP2040 is a dual-core ARM chip developed by Raspberry Pi.

Supported RP2040 devices include:

Raspberry Pi Pico + Waveshare LoRa Module (Note: **Bluetooth on the Pico W is not yet supported by Meshtastic**)

RAK11310 Core module for RAK WisBlock modular boards

 **INFO**

If your device is not listed above, please review our supported devices to determine which MCU your device has or contact us in Discord with any questions.

 **STOP! PUT THE POWER CABLE DOWN!**

Never power on the radio without attaching an antenna! *It* could damage the radio chip.

Prior to connecting your Meshtastic device to the computer, you should perform the following basic checks.

Verify Data Cable

Some cables only provide *charging*, verify that your cable is also capable of *transferring data* before proceeding. To check if your cable can also transfer data, try connecting it to another device (like a phone) and see if you can copy a file to or from it. If the file transfer works, then your cable is also able to transfer data and you can continue.

Install Serial Drivers

CAUTION

nRF52/RP2040 devices typically do not require serial drivers. They use the UF2 bootloader which makes the devices appear as flash drives. Do *NOT* download the USB device drivers unless required to install UF2 support.

If you require serial drivers installed on your computer, please choose one of the options below and install it before continuing.

[Install ESP32 Drivers](#)

Install nRF52/RP2040 Drivers

Flash Firmware

After completing the previous steps, you can now flash the Meshtastic firmware onto your device. To proceed, select the appropriate device type for your device.

Flash ESP32 Firmware

Flash nRF52/RP2040 Firmware

Connect and Configure Device

After flashing the Meshtastic firmware onto your device, you can now move on to initial configuration.

Connect and Configure Device

Installing Serial Drivers



ESP32 Drivers

Install ESP32 USB to Serial Drivers



nRF52/RP2040 Drivers

Install nRF52/RP2040 USB to Serial Drivers



Test Serial Driver Installation

Test Serial Driver Installation

ESP32 Serial Drivers

Install ESP32 USB to Serial Drivers

You may need to install a driver from Silicon Labs for the CP210X USB to UART bridge

Some newer boards may require the CH9102 (CH340/CH341) Driver.

Linux

[CP210X USB to UART bridge - Download](#)

[CH9102 Driver - Linux Download](#)

macOS

[CP210X USB to UART bridge - Download](#)

[CH9102 Driver - MacOS Download](#)

Windows

[CP210X USB to UART bridge - Download](#)

[CH9102 Driver - Windows Download](#)

[CH9102 Driver - Windows Download \(Direct Download for Windows 7\)](#)

⚠️ IMPORTANT

After installing the driver, make sure to reboot your computer to finish the installation process.

You can also test your serial driver installation at this step if

required.

Flash Firmware

After installing the serial drivers, you can now flash the Meshtastic firmware onto your device. To proceed, select the appropriate device type for your device.

[Flash ESP32 Firmware](#)

nRF52/RP2040 Serial Drivers

Install nRF52/RP2040 USB to Serial Drivers

CAUTION

nRF52/RP2040 devices typically do not require serial drivers. They use the UF2 bootloader which makes the devices appear as flash drives. Do *NOT* download the USB device drivers unless required to install UF2 support.

Linux

[CH34x Driver - Linux Download](#)

macOS

INFO

With the latest versions of MacOS, the USB Serial driver is built-in. If you downloaded/installed any already, please remove them.

Remove the CH34x USB Driver (macOS)

If you have already downloaded/installed the macOS WCH-IC CH340/CH341 ("CH341SER_MAC") drivers via the CH34x_Install_V1.5.pkg, you will have to Uninstall the kernel extension:

Unplug your device

Open the Terminal and run:

```
sudo rm -rf /Library/Extensions/usbserial.kext
```

Reboot

Install the CH34x Driver

CH34x Driver- macOS Download

Windows

CH34x Driver - Windows Download

ⓘ IMPORTANT

After installing the driver, make sure to reboot your computer to finish the installation process.

You can also test your serial driver installation at this step if required.

Flash Firmware

After installing the serial drivers, you can now flash the Meshtastic firmware onto your device. To proceed, select the appropriate device type for your device.

Flash nRF52/RP2040 Firmware

Test Serial Driver Installation

Test Serial Driver Installation

You can verify that you have a proper data cable (rather than a charge-only type cable) and that the appropriate drivers for your system are installed by performing the following test:

Linux

Connect your Meshtastic device to your USB port

Open a **Terminal** and enter the following command:

```
lsusb
```

You should see something like:

```
ID xxxx:xxxx Silicon Labs CP210x UART Bridge
# or
ID xxxx:xxxx QinHeng Electronics USB Single Serial
# or
ID xxxx:xxxx Adafruit WisCore RAK4631 Board
```

macOS

Navigate to **Apple Menu ◊ > About This Mac > More Info > System Report... > Hardware > USB.**

You should see similar to one of the following entries:

CP210X USB to UART Bridge Controller

CH9102 USB to UART Bridge Controller

WisCore RAK4631 Board

USB Single Serial

Windows

Navigate to Device Manager > Ports (COM & LPT)

You should see similar to one of the following entries:

Silicon Labs CP210X USB to UART Bridge (COM5)

Silicon Labs CH9102 USB to UART Bridge (COM5)

USB-Enhanced-SERIAL CH9102 (COM5)

USB Serial Device (COM5)

ⓘ INFO

If you are unable to see your device:

Make sure that your cable is not only for charging but also for data transfer.

It's possible that you need to reinstall the USB serial driver.

Flash Firmware

After completing the previous steps, you can now flash the Meshtastic firmware onto your device. To proceed, select the appropriate device type for your device.

Flash ESP32 Firmware

Flash nRF52/RP2040 Firmware

Flashing Firmware



ESP32 Device

3 items



nRF52/RP2040 Device

5 items

Flash ESP32 Devices

 **INFO**

The recommended method for firmware flashing is the Web-Based Installer.

Flashing Method for ESP32 Devices

The Web-Based Installer requires either Chrome or Edge browsers but is an excellent choice for quickly flashing devices. **This method is highly recommended for firmware flashing, especially for new users of the project, as it is easy to use.**

The CLI Script is considered the "manual process" for flashing firmware.

Flashing your device using an external serial adapter should only be attempted as a last resort if no other method has been successful.

 **NOTE**

The web client at meshtastic.local is only updated with a full wipe and reinstall of the device. If you choose a reinstall, you will get the latest (bundled) web interface. To preserve your settings, you may export your configuration prior to a reinstall and load them back after.

Meshtastic Web Flasher

Flash Device

Plug in your device

Visit flasher.meshtastic.org *requires Chrome or Edge browser

Follow the instructions

Connect and Configure Device

After flashing the Meshtastic firmware to the device, you can proceed with the initial configuration.

[Connect and Configure Device](#)

Flashing with the CLI

CAUTION

Make sure not to power the radio on without first attaching the antenna! You could damage the radio chip!

Before you flash your device start by verifying connectivity with the device being flashed. Outlined below are steps that can be taken to verify connectivity and, if necessary, to install the appropriate drivers. If you end up needing to install drivers be sure to reboot your computer afterwards to verify the installation is complete.

NOTE

The T-Beam 0.7 board is an earlier version of the T-Beam board, and due to changes in the design in subsequent iterations this board uses a specific firmware file different from the other T-Beam boards.

`firmware-tbeam0.7-X.X.X.xxxxxxx.bin` is the correct firmware. `firmware-tbeam-X.X.X.xxxxxxx.bin` is incompatible. For all other T-Beam boards `firmware-tbeam-X.X.X.xxxxxxx.bin` is the correct selection.

Command Line Interface Instructions

Install Prerequisite Software

Linux

Check if you have `python3` and `pip` installed with the following command

```
python3 --version  
pip3 --version
```

If `python3` is not installed, install with

```
sudo apt-get update  
sudo apt-get install python3
```

If `pip` is not installed, install with

```
sudo apt-get install python3-pip
```

macOS

OS X comes with `Python 2.7` installed, but not `pip`. The following uses Homebrew to install `python3` which includes `pip3`. On MacOS you will use `pip3` instead > of `pip`.

NOTE

Check if you have Homebrew installed with the following

command

```
brew -v
```

If it's not installed, follow the instructions on the Homebrew website before continuing.

Check if you have `python3` and `pip` installed with the following command

```
python3 --version  
pip3 --version
```

If `python3` is not installed, install with Install Python3

```
brew install python3
```

Confirm `pip3` was installed alongside `python3`

```
pip3 -v
```

Windows

Download and install Python. When installing, make sure to click `Add Python X.Y to PATH`.

Download and install Gitbash (or other appropriate shell) and run all subsequent commands from that shell.

 **NOTE**

Confirm installation of `python` & `pip` with the following commands.

```
py --version
```

```
pip --version
```

Install esptool

```
pip3 install --upgrade esptool
```

Confirm Communication With Chip

Linux

! IMPORTANT

On Linux, you may need to explicitly declare esptool as a .py script. Use `esptool.py chip_id`.

macOS

! IMPORTANT

On macOS, you may need to explicitly declare esptool as a .py script. Use `esptool.py chip_id`.

Windows

! IMPORTANT

On Windows, you must explicitly declare esptool as a .py script. Use `esptool.py chip_id`.

Connect the radio to your computer using a data USB cable. Confirm your device is talking to your computer using the following command:

Command

```
esptool chip_id
```

Expected Output

```
# You should see a result similar to this:  
mydir$ esptool chip_id  
esptool.py v2.6  
Found 2 serial ports  
Serial port /dev/ttyUSB0  
Connecting....  
Detecting chip type... ESP32  
Chip is ESP32D0WDQ6 (revision 1)  
Features: WiFi, BT, Dual Core, 240MHz, VRef  
calibration in efuse, Coding Scheme None  
MAC: 24:6f:28:b5:36:71  
Uploading stub...  
Running stub...  
Stub running...
```

Navigate to Firmware

`cd` into the directory where you unzipped the latest release. For example:

Example

```
cd ~/Downloads/firmware/
```

Install/Update Firmware

Install or Update the device that you have by using the following commands according to your operating system:

⚠ CAUTION

Be very careful to install the correct firmware file for your board. In particular, the popular 'T-BEAM' radio from TTGO is not called 'TTGO-Lora' (that is a different board). So don't install the 'TTGO-Lora' build on a TBEAM, it won't work correctly. If you flash the incorrect device firmware, it may appear to successfully complete, yet the device will usually become unresponsive, displaying a blank screen (if equipped). Attempt to flash the correct version. Please note, while these devices are quite resilient and damage is not always likely, there is a possibility of onboard radio and peripheral damage

due to incorrect GPIO pin direction.

Linux **Install**

Command

```
./device-install.sh -f firmware-BOARD-VERSION.bin
```

Update

Command

```
./device-update.sh -f firmware-BOARD-VERSION.bin
```

macOS **Install**

Command

```
./device-install.sh -f firmware-BOARD-VERSION.bin
```

Update

Command

```
./device-update.sh -f firmware-BOARD-VERSION.bin
```

Windows **Install**

Command

```
device-install.bat -f firmware-BOARD-VERSION.bin
```

Update

Command

```
device-update.bat -f firmware-BOARD-VERSION.bin
```

Connect and Configure Device

After flashing the Meshtastic firmware to the device, you can proceed with the initial configuration.

[Connect and Configure Device](#)

Flashing with an External Serial Adapter

ⓘ INFO

This information will likely only be helpful if you've already attempted to go through the prerequisites and processes outlined in manually flashing

⚠ CAUTION

Make sure not to power the radio on without first attaching the antenna! You could damage the radio chip!

Background

Situations that may require using an external USB to Serial Adapter:

Due to the chip shortage, recently purchased devices such as the TTGO T-Beam may come with legacy or non-standard USB to Serial adapter chips that are unreliable in some cases.

Certain devices might have defective USB to Serial chip.

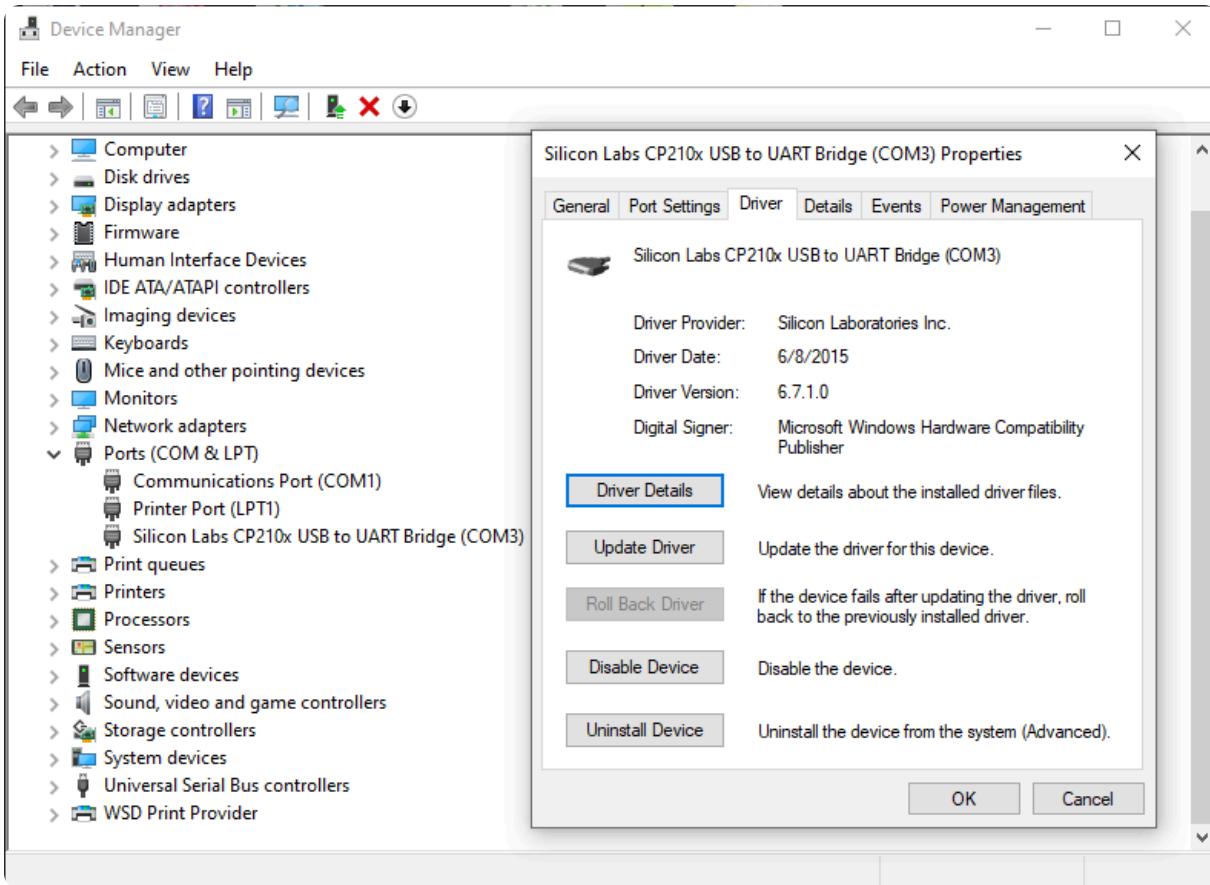
Certain devices, such as the Hydra (Meshtastic-DIY target).

USB Serial Adapters

There are many options on the marketplace. It is recommended to purchase an adapter featuring the Silicon Labs CP2102 chip, as it is a reliable industry standard.

The adapter featured in this document can be purchased from:
<https://www.amazon.com/gp/product/B078W5L8W1/>

Plug the adapter into your computer without connecting it to any devices yet. Ensure that your computer has drivers installed for the adapter. On Windows, the correct drivers should install automatically.



Connecting Adapter to the Device

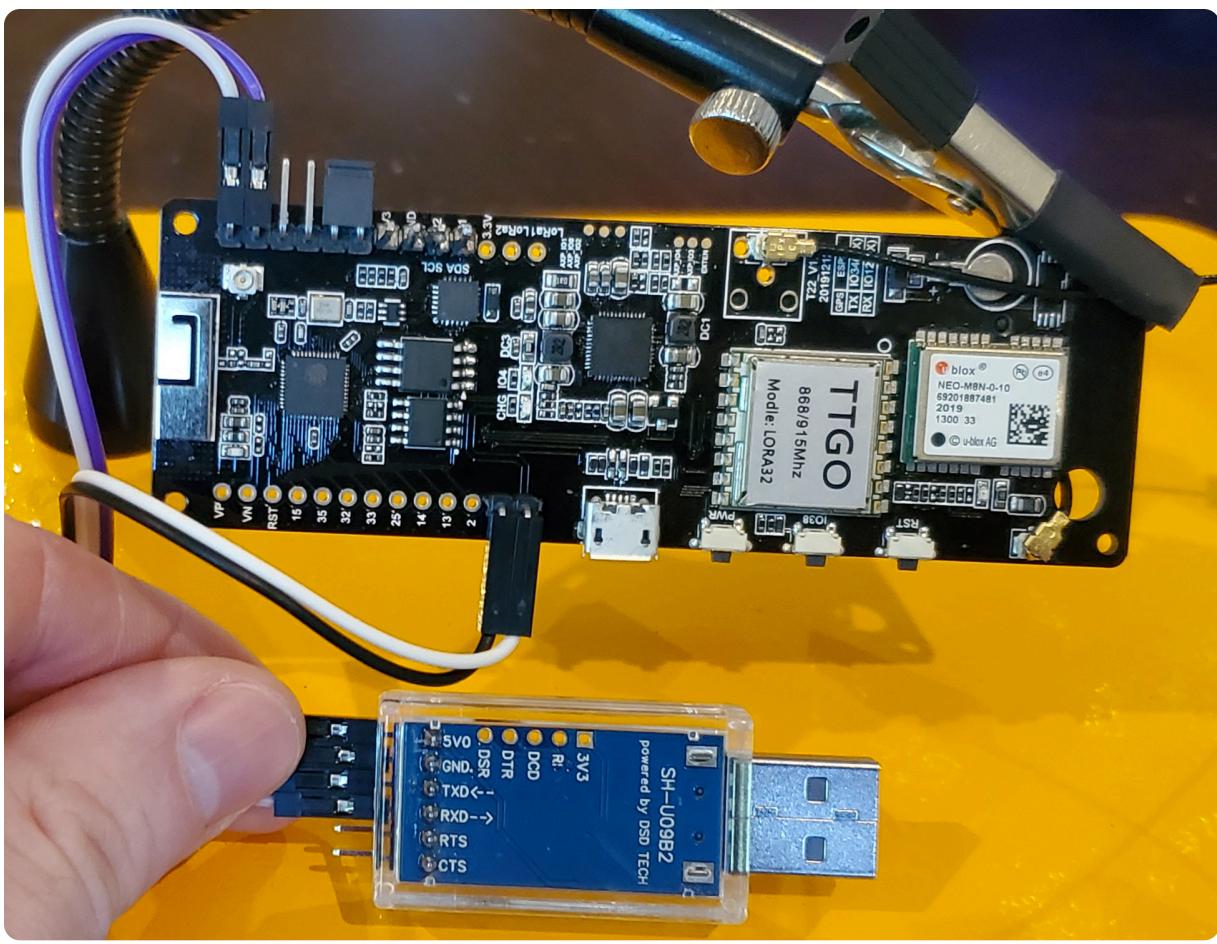
INFO

There are multiple ways to connect the pins of the adapter to the target device: pressing jumpers against contacts, using pogo pin jigs, etc. This tutorial features offset dupont headers soldered onto the operative GPIO pins and connected via jumpers.

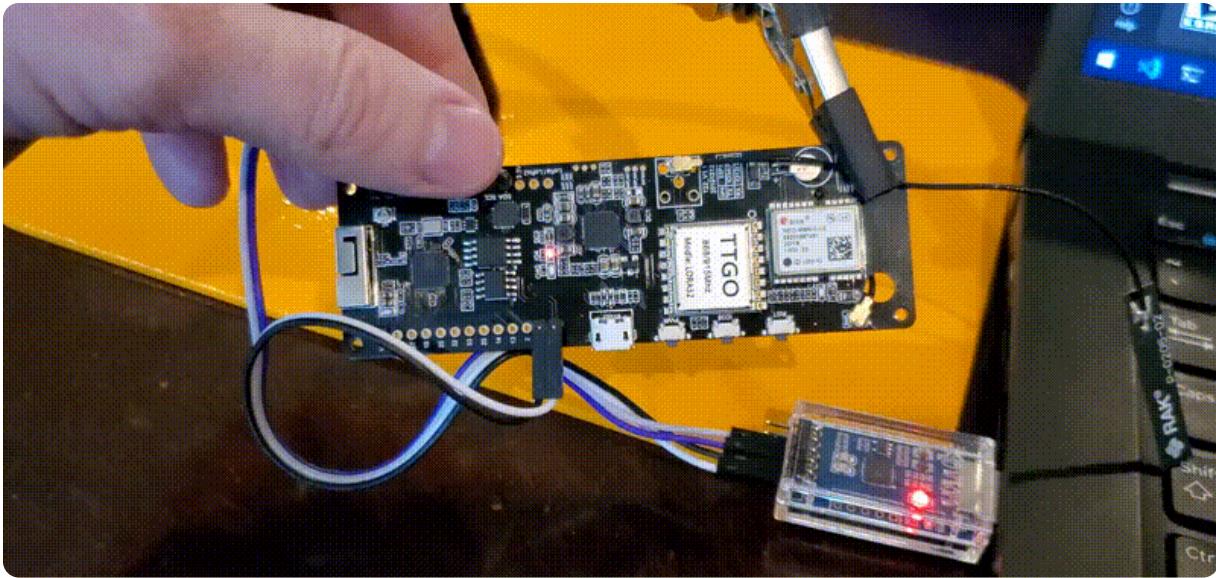
Disconnect your USB to Serial Adapter from the computer before starting this process.

Connect the RX pin of the adapter to the TX pin of the device
Connect the TX pin of the adapter to the RX pin of the device
Connect a GND pin of the adapter to the GND pin of the device
Connect either the 5V pin of the adapter to the 5V pin of the device (illustrated) or the 3.3V pin of the adapter to the 3.3V pin of device.

Bridge GPIO 0 to GND on the device with a jumper. (This places the device into flash mode when the device is powered up) Example wiring featuring a T-Beam



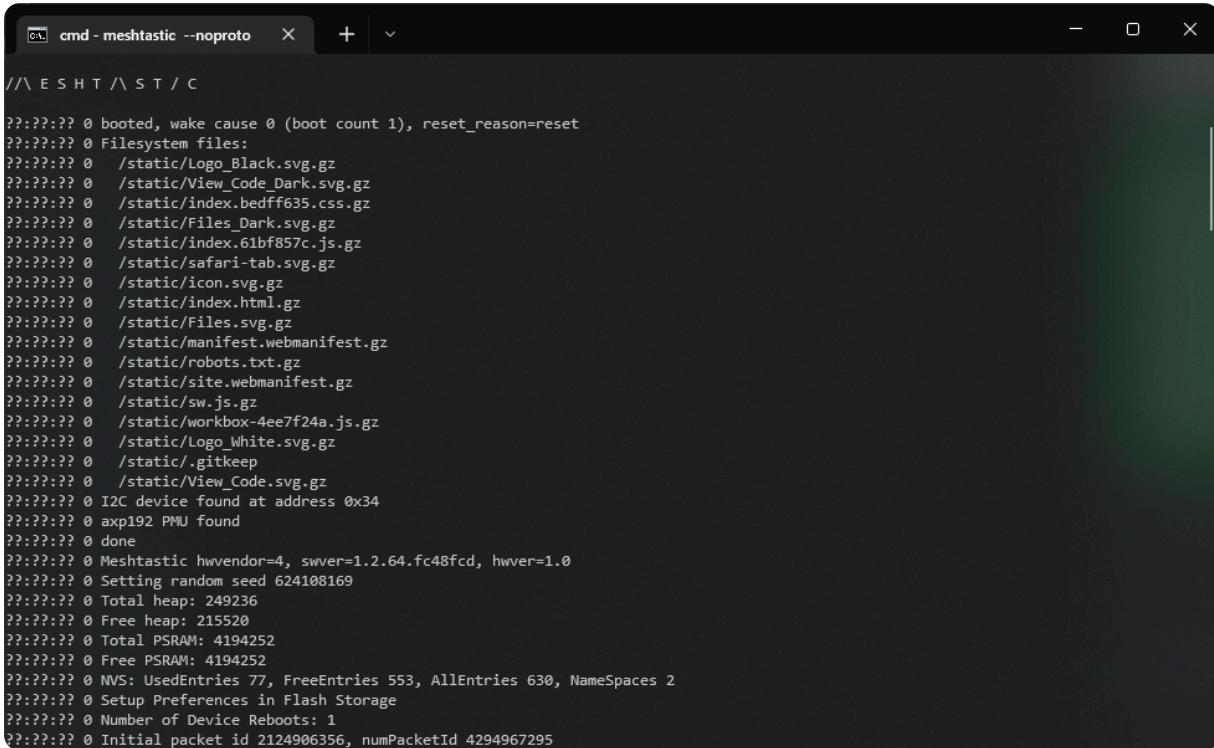
Connect the device to a USB port on the computer
Remove the jumper bridging GPIO 0 to GND



Flashing

After following the steps above, your device should be in flash mode. You can flash your device using the manual method

After flashing the device is complete, reset your device (via the RST button if available). If you have the Meshtastic Python CLI installed, you can run `meshtastic --noprotocol` to connect the device again over the adapter and view the serial output to confirm Meshtastic installed correctly.



```
//\ E S H T /\ S T / C

???:???:? 0 booted, wake cause 0 (boot count 1), reset_reason=reset
???:???:? 0 Filesystem files:
???:???:? 0  /static/Logo_Black.svg.gz
???:???:? 0  /static/View_Code_Dark.svg.gz
???:???:? 0  /static/index.bedff635.css.gz
???:???:? 0  /static/Files_Dark.svg.gz
???:???:? 0  /static/index.01bf857c.js.gz
???:???:? 0  /static/safari-tab.svg.gz
???:???:? 0  /static/icon.svg.gz
???:???:? 0  /static/index.html.gz
???:???:? 0  /static/Files.svg.gz
???:???:? 0  /static/manifest.webmanifest.gz
???:???:? 0  /static/robots.txt.gz
???:???:? 0  /static/site.webmanifest.gz
???:???:? 0  /static/sw.js.gz
???:???:? 0  /static/workbox-4ee7f24a.js.gz
???:???:? 0  /static/Logo_White.svg.gz
???:???:? 0  /static/.gitkeep
???:???:? 0  /static/View_Code.svg.gz
???:???:? 0 I2C device found at address 0x34
???:???:? 0 axp192 PMU found
???:???:? 0 done
???:???:? 0 Meshtastic hwvendor=4, swver=1.2.64.fc48fcfd, hwver=1.0
???:???:? 0 Setting random seed 624108169
???:???:? 0 Total heap: 249236
???:???:? 0 Free heap: 215520
???:???:? 0 Total PSRAM: 4194252
???:???:? 0 Free PSRAM: 4194252
???:???:? 0 NVS: UsedEntries 77, FreeEntries 553, AllEntries 630, NameSpaces 2
???:???:? 0 Setup Preferences in Flash Storage
???:???:? 0 Number of Device Reboots: 1
???:???:? 0 Initial packet id 2124906356, numPacketId 4294967295
```

Troubleshooting

You might receive an error for COM port permission in the manual device install scripts, this can be caused by a number of different issues.

You might need to run the process as an administrator, check to ensure software like Cura isn't monopolizing COM ports, or reboot.

Connect and Configure Device

After flashing the Meshtastic firmware to the device, you can proceed with the initial configuration.

[Connect and Configure Device](#)

Flash nRF52 & RP2040 Devices

Flashing Methods for nRF52 and RP2040 Devices

nRF52 and RP2040 based devices have the easiest firmware upgrade process. No driver or software install is required on any platform.

Drag & Drop

nRF52 and RP2040 devices use the Drag & Drop installation method to install firmware releases.

Over-The-Air (OTA)

nRF52 devices are able to accept OTA firmware updates from a mobile device over bluetooth.

nRF Factory Erase

You may wish to perform a Factory Erase prior to installing firmware to clear data that may change format and location between releases.

Convert RAK4631-R to RAK4631

If your device did not come with the Arduino bootloader you will need to perform the conversion.

Drag & Drop nRF52 & RP2040 Firmware Updates

Flash Firmware

 **INFO**

You may now use the Meshtastic Web Flasher to download and copy firmware to your nRF52 or RP2040-based devices.

Alternatively, follow the instructions below to download and install firmware.

nRF52

Download and unzip the latest firmware from Meshtastic Downloads.

Connect your device to your computer with a USB data cable.

Double click the reset button on your device (this will put it into bootloader mode).

Notice a new drive will be mounted on your computer (Windows, Mac, Linux, or Android).

Open this drive and you should see three files: `CURRENT.UF2`, `INDEX.HTM`, and `INFO_UF2.TXT`.

Copy the appropriate firmware file (`firmware-DEVICE_NAME-`

`X.X.X-xxxxxxxx.uf2`) from the release onto this drive.

Once the file has finished copying onto the drive, the device will reboot and install the Meshtastic firmware.

RP2040

Download and unzip the latest firmware from Meshtastic Downloads.

Press the BOOTSEL button and while keeping it pressed, connect the device to your computer via a USB cable.

Notice a new drive will be mounted on your computer (Windows, Mac, Linux, or Android).

Open this drive and you should see two files: `INDEX.HTM` and `INFO_UF2.TXT`.

Copy the appropriate firmware file (`firmware-DEVICE_NAME-X.X.X-xxxxxxxx.uf2`) from the release onto this drive.

Once the file has finished copying onto the drive, the device will reboot and install the Meshtastic firmware.

Connect and Configure Device

After flashing the Meshtastic firmware to the device, you can proceed with the initial configuration.

[Connect and Configure Device](#)

Potential Flashing Edge Cases

ⓘ INFO

Before flashing confirm that you have RAK4631 and not a RAK4631-R. If this is not the case, fear not. The hardware is identical but requires changing the bootloader. Instructions on how to do this are located [here](#).

ⓘ INFO

Previous versions of the Meshtastic firmware may save stale data, causing devices to get stuck in a crash loop during startup. If you experience issues when upgrading your nRF52 device from a previous version of Meshtastic, you may need to perform a full factory reset of the internal flash memory.

Follow the guide to factory erase your nRF52 device before continuing to flash firmware.

nRF52 OTA Firmware Updates

nRF52 devices from RAK are able to accept OTA firmware updates from a mobile device over bluetooth. Older T-Echo bootloaders do not have OTA support.

Android

ⓘ INFO

As of this writing, the current Android release of the nRF DFU app (v2.3.0) is not compatible with Meshtastic firmware updates. Please use the instructions below for updating via OTA with the nRF Connect App.

OTA firmware updates are available for Android using an older release of the more advanced nRF Connect App **version 4.24.3** which is available for download from the Nordic Semiconductor GitHub page.

Download the firmware release you wish to install from the Meshtastic Download Page or Meshtastic GitHub.

Unzip the firmware folder

Open the nRF Connect App and select CONNECT on your device from the SCANNER tab

Select the DFU icon from the top-right of the screen

Select the correct device firmware file (will end with -ota.zip)

The update will start automatically

Apple

OTA firmware updates are available on iOS & iPadOS using the nRF Device Firmware Update App available through the Apple App Store

Download the firmware release you wish to install from the Meshtastic Download Page, Meshtastic GitHub, or via the iOS or iPadOS app.

Unzip the firmware folder

Open the nRF DFU App and select the correct device firmware file (will end with -ota.zip)

Connect to your device

Upload the firmware

Flash nRF52/RP2040 Factory Erase

Meshtastic uses the littlefs library to store configuration, logs, and other data in the internal flash of nRF52 & RP2040 devices.

Updating the firmware does *not* erase this additional data, which can cause issues when the format and location of data changes between releases.

ⓘ INFO

You may now use the Meshtastic Web Flasher to Factory Erase your nRF52 or RP2040-based devices. Visit the flasher, select your board, and click the trash can icon to the right of the Flash button. This will open a dialogue to begin the erase procedure.

Alternatively, follow the instructions below.

nRF52

To reset the flash storage on your nRF52 board:

Download and unzip the latest firmware from Meshtastic Downloads.

Connect your device to your computer with a USB data cable.

Double click the reset button on your device (this will put it into

bootloader mode)

Notice a new drive will be mounted on your computer (Windows, Mac, or Linux)

Open this drive and you should see three files: CURRENT.UF2, INDEX.HTM, and INFO_UF2.TXT

Find the file included in the downloaded firmware named Meshtastic_nRF52_factory_erase.uf2 and copy it onto the new drive. The device should reboot.

With the Meshtastic CLI installed, open your Terminal/Console and enter the command: meshtastic --noprotocol.

Press any key, you should see the message: Formatting...

Done.

Once the device has been erased, you can proceed to install the latest Meshtastic firmware on a clean storage filesystem by clicking the link below.

RP2040

To reset the flash storage on your RP2040 board:

Download flash_nuke.uf2 to your computer.

Press the BOOTSEL button and while keeping it pressed, connect it to your computer via a USB cable.

The board should now appear as a mass storage device on your computer with the label RPI-RP2.

Copy the flash_nuke.uf2 file to the device.

The board will now restart and the flash memory will be erased.

Once the device has been erased, you can proceed to install the latest Meshtastic firmware on a clean storage filesystem by clicking the link below.

[**Flash nRF52/RP2040 Firmware**](#)

Convert RAK4631-R to RAK4631

The only difference between the *RAK4631-R* (RUI3) and the *RAK4631* (Arduino) is the bootloader it is shipped with - the hardware is the same.

Meshtastic requires the Arduino bootloader on RAK WisBlock nRF52-based boards. The process of converting the bootloader only needs to be performed once.

Here are two ways to flash the bootloader:

USB Device Firmware Upgrade (DFU)

Install Python

Install adafruit-nrfutil

```
pip3 install adafruit-nrfutil
```

Download the required bootloader:

WisCore_RAK4631_Board_Bootloader.zip

Connect your RAK device by USB.

Flash the bootloader

```
adafruit-nrfutil --verbose dfu serial --package  
./WisCore_RAK4631_Board_Bootloader.zip -p /dev/  
ttyACM0 -b 115200 --singlebank --touch 1200
```

Note: The serial port name (`/dev/ttyACM0`) may differ depending on your operating system. Make sure to identify the correct port name for your setup.

Continue with the normal flashing instructions

Debugger

This conversion requires the use of either a DAPLink or J-Link. The most reasonably priced and available is the RAKDAP1.

Install Python

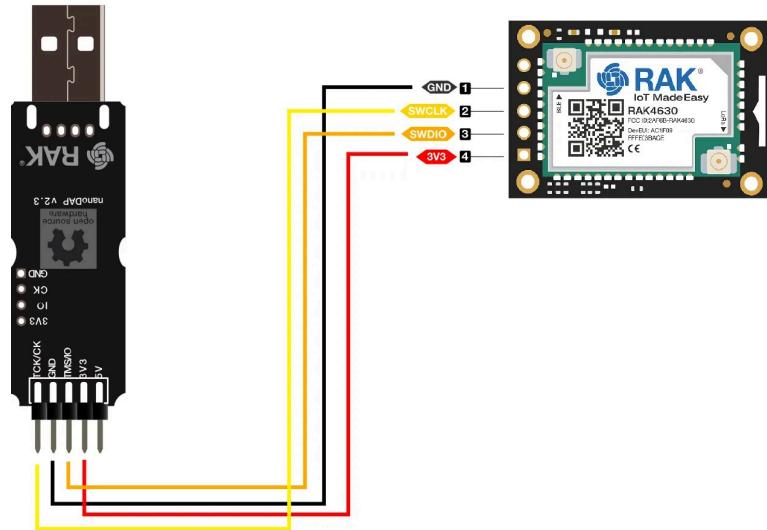
Install pyOCD

```
pip3 install pyocd
```

Download the required bootloader:

WisCore_RAK4631_Board_Bootloader.hex

Connect the RAKDAP as follows:



Flash the bootloader

```
pyocd flash -t nrf52840
.\WisCore_RAK4631_Board_Bootloader.hex
```

Continue with the normal flashing instructions

Alternate methods of flashing are outlined here.

How to Update the LilyGo T-Echo Bootloader to the Latest Version

If you're experiencing issues with updating or flashing newer versions of the Meshtastic firmware, and your LilyGo T-Echo is not running the latest bootloader version (0.6.1), updating the bootloader may resolve these problems.

To check which version of the bootloader your device is running, place the device into DFU mode by double-pressing the reset button. Then, open the mounted drive that appears on your computer and check the INFO_UF2.TXT file.

Updating bootloader

Below are the steps to update your bootloader.

Method 1: Using the UF2 File (Recommended)

Download the Latest UF2 Bootloader File for the T-Echo from Github.

Connect your LilyGo T-Echo to your computer via USB. Activate bootloader mode by quickly double pressing the RESET button on your T-Echo. The device should appear as a removable drive on your computer.

Drag and drop the .uf2 file you downloaded into the T-Echo's drive. The device will automatically update the bootloader and reset.

Once the device resets, the update is complete. Your T-Echo is now running the latest bootloader version and you can proceed with flashing the firmware.

Method 2: Using adafruit-nrfutil

Should flashing the UF2 file to update your bootloader fail, you can use adafruit-nrfutil.

 **INFO**

These instructions assume you have python and pip already installed. If you do not, please install the latest verion of python (which includes pip) from Python.org.

Open a terminal or command prompt and install adafruit-nrfutil by running:

```
pip install adafruit-nrfutil
```

Obtain the lilygo_techo_bootloader-0.6.1.zip package from Github. Connect your LilyGo T-Echo to your computer via USB. In the terminal or command prompt, navigate to the directory

where you downloaded the bootloader zip package and execute the following command, replacing /dev/ttyACM0 with the correct port for your device (Windows users might use COMx):

```
adafruit-nrfutil --verbose dfu serial --package  
lilygo_techo_bootloader-0.6.1.zip -p /dev/ttyACM0 -b  
115200 --singlebank --touch 1200
```

Once the process finishes, the update is complete. Your T-Echo is now running the latest bootloader version and you can proceed with flashing the firmware.

Initial Configuration

Supported Clients per Connection Type

Depending on your connection, some configuration options are not fully supported. Find out which client is best for your type of connection.

Serial

Python CLI

Web Client

Android App

Bluetooth

Android App

Web Client

iOS/iPadOS/macOS App

Network



INFO

Connecting over network is only supported on ESP32 devices.

Web Client

Android App

iOS App

Python CLI

Set Regional Settings

In order to start communicating over the mesh, you must set your region. This setting controls which frequency range your device uses and should be set according to your regional location.

Android

Follow the installation and usage instructions for Meshtastic Android.

Open the app, connect to the device from your phone over USB Serial or Bluetooth.

Once paired, Click "UNSET" next to the device name.

Select the region from the list according to your regional location.

Apple

ⓘ INFO

Configuration of Region, Modem Preset and Hop Limit is available on iOS, iPadOS and macOS at Settings > Radio Configuration > LoRa.

CLI

Install Meshtastic PythonCLI

```
pip3 install --upgrade pytap2  
pip3 install --upgrade meshtastic
```

Run the following command, replacing <REGION-CODE> with the region code listed above according to your regional location.

```
meshtastic --set lora.region <REGION-CODE>
```

Web

Open the Meshtastic Web interface: client.meshtastic.org

Navigate to the **LoRa** menu.

Under **Regional Settings**, set your **Region** according to your regional location.

Click **Save**.

Region Codes

Region Code	Description
UNSET	Unset
US	United States
EU_433	European Union 433MHz
EU_868	European Union 868MHz
CN	China
JP	Japan
ANZ	Australia & New Zealand

Region Code	Description
KR	Korea
TW	Taiwan
RU	Russia
IN	India
NZ_865	New Zealand 865MHz
TH	Thailand
UA_433	Ukraine 433MHz
UA_868	Ukraine 868MHz
MY_433	Malaysia 433MHz
MY_919	Malaysia 919MHz
SG_923	Singapore 923MHz
LORA_24	2.4 GHz band worldwide

 **INFO**

EU_433 and EU_868 have to adhere to an hourly duty cycle limitation of 10%. Your device will stop transmitting if you reach it, until it is allowed again.

Continue Configuration

Now that you have set the LoRa region on your device, you can continue with configuring any additional configs to suit your needs.

[Device Configuration](#)

Configuration

Radio Config

9 items

Module Config

14 items

Remote Nodes

An advanced feature which allows remote administration of a device through a se...

Tips

Tips and Solutions to help you get the most out of your Meshtastic device and net...

Radio Configuration

There are several config sections in the Meshtastic firmware, these are broken out so they can be sent as small admin messages over the mesh.

Name	Description
Bluetooth	Bluetooth config options are: Enabled, Pairing Mode and Fixed PIN.
Channels	Channels config options are: Index, Role and Settings.
Device	Device config options are: Device Role, Serial Output, Debug Log and Factory Reset.
Display	Display config options are: Screen On Duration, Auto Carousel Interval, Always Point North, and GPS Format.
LoRa	The LoRa config options are: Region, Modem Preset, Max Hops, Transmit Power, Bandwidth, Spread Factor, Coding Rate, Frequency Offset, Transmit Disabled and Ignore Incoming Array.
Network	Network config options are: WiFi Enabled, WiFi SSID,

Name	Description
	WiFi PSK, WiFi Mode and NTP Server.
Position	Position config options are: GPS Enabled, GPS Update Interval, GPS Attempt Time, Fixed Position, Smart Broadcast, Broadcast Interval and Position Packet Flags.
Power	Power config options are: Charge Current, Power Saving, Shutdown after losing power, ADC Multiplier Override Wait Bluetooth Interval, Light Sleep Interval and Minimum Wake Interval.
User	The user config options are: Long Name, Short Name, and Is Licensed

Bluetooth Settings

The Bluetooth config options are: Enabled, Pairing Mode and Fixed PIN Value. Bluetooth config uses an admin message sending a `Config.Bluetooth` protobuf.

 **INFO**

ESP32 Devices: Bluetooth will be disabled if WiFi is enabled.
The WiFi setting takes precedence.

Bluetooth Config Values

Enabled

Enables Bluetooth.

Pairing Mode

Specify pairing mode.

`RANDOM_PIN` generates a random PIN during runtime. `FIXED_PIN` uses the fixed PIN that should then be additionally specified.
Finally, `NO_PIN` disables PIN authentication.

Default Pairing Mode

The default pairing mode will be determined based on whether the device has or does not have a screen attached to it during the first boot (or with a stale device state) unless manually configured via the Bluetooth config options.

Screen Attached: If your device boots up for the first time (or with a stale device state), and a screen is detected, the default pairing mode will be set to `RANDOM_PIN`. Should the attached screen be removed after the device has already been booted, the default pairing mode of `RANDOM_PIN` will remain unless manually changed to `FIXED_PIN` or `NO_PIN`. It is recommended the pairing mode be updated prior to removing the attached screen.

No Screen Attached: If your device boots up for the first time (or with a stale device state), and no screen is detected, the default pairing mode will be set to `FIXED_PIN` with the default value listed below unless manually configured to a custom value.

Fixed PIN

If your pairing mode is set to `FIXED_PIN`, the default value is **123456**. For all other pairing modes, this number is ignored. A custom integer (6 digits) can be set via the Bluetooth config options.

WARNING

It is strongly recommended that you change the default **FIXED_PIN** code on your device. Leaving the default code in place can pose a significant security risk.

Bluetooth Config Client Availability

Android

INFO

All Bluetooth config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Bluetooth**

Apple

INFO

All bluetooth config options are available on iOS, iPadOS and macOS at Settings > Device Configuration > Bluetooth.

CLI

INFO

All Bluetooth config options are available in the python CLI.

Example commands are below:

Setting	Acceptable Values	Default
bluetooth.enabled	true, false	true
bluetooth.mode	RANDOM_PIN, FIXED_PIN, NO_PIN	RANDOM_PIN
bluetooth.fixedPin	integer (6 digits)	123456

 **TIP**

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set bluetooth.enabled true --set
bluetooth.fixed_pin 111111
```

Enable/Disable Bluetooth Module

```
meshtastic --set bluetooth.enabled true
meshtastic --set bluetooth.enabled false
```

Set a fixed pin

```
meshtastic --set bluetooth.mode FIXED_PIN  
meshtastic --set bluetooth.fixed_pin 111111
```

Web



INFO

All Bluetooth module config options are available for the Web UI.

Open the Meshtastic Web UI.

Navigate to: **Radio Config > Bluetooth**

Channel Configuration

The Channels config options are: Index, Roles, and Settings.

Channel config uses an admin message sending a `Channel` protobuf which also consists of a `ChannelSettings` or `ModuleSettings` protobuf.

 **INFO**

Channel Settings (as described on this page) should not be confused with Modem Preset Settings

Modem Preset Settings contain the modem configuration (frequency settings, spreading factor, bandwidth, etc.) used for the LoRa radio. These settings are identical for all channels and can **not** be unique per channel.

Channel Settings contain information for segregating message groups, configuring optional encryption, and enabling or disabling messaging over internet gateways. These settings **are** unique and configurable per channel.

Channel Config Values

Index

The channel index begins at 0 and ends at 7.

Indexing can not be modified.

Index	Channel	Default Role	Purpose
0	1	PRIMARY	Used as default channel
1	2	DISABLED	User defined
2	3	DISABLED	User defined
3	4	DISABLED	User defined
4	5	DISABLED	User defined
5	6	DISABLED	User defined
6	7	DISABLED	User defined
7	8	DISABLED	User defined

 **NOTE**

You can **not** have DISABLED channels in-between active channels such as PRIMARY and SECONDARY. Active channels must be consecutive.

Role

Each channel is assigned one of 3 roles:

PRIMARY or **1**

This is the first channel that is created for you on initial setup.

Only one primary channel can exist and can not be disabled.

Periodic broadcasts like position and telemetry are only sent over this channel.

SECONDARY or **2**

Can modify the encryption key (PSK).

DISABLED or **0**

The channel is no longer available for use.

The channel settings are set to default.

NOTE

While you can have a different PRIMARY channel and communicate over SECONDARY channels with the same Name & PSK, a hash of the PRIMARY channel's name sets the LoRa channel number, which determines the actual frequency you are transmitting on in the band. To ensure devices with different PRIMARY channel name transmit on the same frequency, you must explicitly set the LoRa channel number.

Channel Settings Values

The Channel Settings options are: Name, PSK, Downlink Enabled, and Uplink Enabled. Channel settings are embedded in the `Channel` protobuf as a `ChannelSettings` protobuf and sent as an admin message.

Name

A short identifier for the channel. (*< 12 bytes*)

Reserved Name	Purpose
<code>""</code> (default)	If left empty on the Primary channel, this designates the <code>default</code> channel.
<code>admin</code>	On Secondary channels, the name <code>admin</code> (case sensitive) designates the <code>admin</code> channel used to administer nodes over the mesh

NOTE

Matching channel names are required in order to communicate on the same channel with other devices. Example: If your

device is using the channel name `LongFast` the device you are attempting to communicate with must also have a channel named `LongFast`.

PSK

The encryption key used for private channels.

Hex byte `0x01` for the Primary `default` channel.

Must be either 0 bytes (no crypto), 16 bytes (AES128), or 32 bytes (AES256).

 **NOTE**

Matching PSKs are required in order to communicate on the same channel with other devices. Example: If your device is using a channel with the default PSK of `AQ==` the device you are attempting to communicate with must also have a matching channel with the same PSK.

Downlink Enabled

If enabled, messages captured from a **public** internet gateway will be forwarded to the local mesh.

Set to `false` by default for all channels.

Uplink Enabled

If enabled, messages from the mesh will be sent to the **public** internet through any node's configured gateway.

Set to `false` by default for all channels.

Channel Module Settings

The channel module settings options are: position precision. Channel module settings are embedded in the Channel protobuf as a ModuleSettings protobuf and sent as an admin message.

Position Precision

The `position_precision` setting allows control of the level of precision for location data that is sent over a particular channel. This can be useful for privacy reasons, where obfuscating the exact location may be desired when sending position data over certain channels.

The `position_precision` value is an integer between 0 and 32:

A value of 0 means that location data is never sent over the given channel.

A value of 32 means that location data is sent with full precision. Values in between indicate the number of bits of precision to be

sent.

Some useful values and their approximate precisions:

11: Large region, around ± 11 kilometers

13: City-sized region, around ± 3 kilometers

16: Neighborhood-level precision, around ± 350 meters

The client applications have implemented different levels of precision giving the user a practical range to choose from. Setting across the full range of integers can be done via the Python CLI. See [Setting Position Precision](#) for examples on setting different levels of precision using CLI.

Channel Config Client Availability

Android



INFO

Channel Config options are available on Android.



The Radio Configuration tab can be used for common tasks:

View your current channel configuration QR code and URL.

Quickly create or modify your primary channel.

Select a modem preset for all your channels i.e. **Long Range /**

Fast.

See [Android App Usage](#) for more further instruction on setting up

your primary channel.

Channel Name
#LongFast-I



Tap the Channel Name (or the pen icon) to access the Channel Menu:

Add, remove, or modify secondary channels

Create or modify encryption keys

Enable uplink and downlink for individual channels

Apple

ⓘ INFO

A channel editor is available on the iOS, iPadOS and macOS applications at Settings > Radio Configuration > Channels.

CLI

ⓘ INFO

All Channel config options are available in the python CLI.

Example commands are below:

💡 TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --ch-set name "My Channel" --ch-set
```

Name

Set channel name for the PRIMARY channel

```
# without spaces  
meshtastic --ch-set name MyChannel --ch-index 0  
# with spaces  
meshtastic --ch-set name "My Channel" --ch-index 0
```

PSK

If you use Meshtastic for exchanging messages you don't want other people to see, `random` is the setting you should use.

Selecting `default` or any of the `simple` values from the following table will use publicly known encryption keys. They're shipped with Meshtastic source code and thus, anyone can listen to messages encrypted by them. They're great for testing and public channels.

Setting	Behavior
<code>none</code>	Disable Encryption
<code>default</code>	Default Encryption (use the weak encryption key)
<code>random</code>	Generate a secure 256-bit encryption key. Use this setting for private communication.
<code>simple0</code> - <code>simple254</code>	Uses a single byte encoding for encryption

Set encryption to default on PRIMARY channel

```
meshtastic --ch-set psk default --ch-index 0
```

Set encryption to random on PRIMARY channel

```
meshtastic --ch-set psk random --ch-index 0
```

Set encryption to single byte on PRIMARY channel

```
meshtastic --ch-set psk simple15 --ch-index 0
```

Set encryption to your own key on PRIMARY channel

```
meshtastic --ch-set psk
```

Set encryption to your own key on PRIMARY channel (Base64 encoded)

```
meshtastic --ch-set psk  
base64:puavdd7vtYJh8NUVWgxbsoG2u9Sdqc54YvMLS+KNcMA=  
--ch-index 0
```

TIP

Use this to copy and paste the `base64` encoded (single channel) key from the `meshtastic --info` command. Please don't use the omnibus (all channels) code here, it is not a valid key.

Disable encryption on PRIMARY channel

```
meshtastic --ch-set psk none --ch-index 0
```

Uplink / Downlink

For configuring gateways, please see MQTT

Enable/Disable Uplink on PRIMARY channel

```
meshtastic --ch-set uplink_enabled true --ch-index 0  
meshtastic --ch-set uplink_enabled false --ch-index 0
```

Enable/Disable Downlink on SECONDARY channel

```
meshtastic --ch-set downlink_enabled true --ch-index 1  
meshtastic --ch-set downlink_enabled false --ch-index 5
```

Setting Position Precision

ⓘ INFO

This is a per-channel setting. The `--ch-index` parameter must be specified to set the position precision for a specific channel, e.g., `--ch-index 0` for the primary channel or `--ch-index 1` for the secondary channel 1.

Set position precision to 13 bits (approx ±3 km)

```
meshtastic --ch-set  
module_settings.position_precision 13 --ch-index 0
```

Set position precision to full precision (32 bits)

```
meshtastic --ch-set  
module_settings.position_precision 32 --ch-index 1
```

Web

ⓘ INFO

All Channel config options are available in the Web UI.

Device Configuration

The device config options are: Role, Serial Output, and Debug Log. Device config uses an admin message sending a `Config.Device` protobuf.

Device Config Values

Roles

Device Role	Description	Best Uses
CLIENT	App connected or stand alone messaging device.	General use for individuals needing to communicate over the Meshtastic network with support for client applications.
CLIENT_MUTE	Device that does not forward packets from other devices.	Situations where a device needs to participate in the network without

Device Role	Description	Best Uses
		assisting in packet routing, reducing network load.
CLIENT_HIDDEN	Device that only broadcasts as needed for stealth or power savings.	Use in stealth/hidden deployments or to reduce airtime/power consumption while still participating in the network.
TRACKER	Broadcasts GPS position packets as priority.	Tracking the location of individuals or assets, especially in scenarios where timely and efficient location updates are critical.
LOST_AND_FOUND	Broadcasts location as message to default channel regularly for to assist with device	Used for recovery efforts of a lost device.

Device Role	Description	Best Uses
	recovery.	
SENSOR	Broadcasts telemetry packets as priority.	Deploying in scenarios where gathering environmental or other sensor data is crucial, with efficient power usage and frequent updates.
TAK	Optimized for ATAK system communication, reduces routine broadcasts.	Integration with ATAK systems (via the Mesthastic ATAK Plugin) for communication in tactical or coordinated operations.
TAK_TRACKER	Enables automatic TAK PLI broadcasts and reduces routine broadcasts.	Standalone PLI integration with ATAK systems for communication in

Device Role	Description	Best Uses
		tactical or coordinated operations.
REPEATER	Infrastructure node for extending network coverage by relaying messages with minimal overhead. Not visible in Nodes list.	Best positioned in strategic locations to maximize the network's overall coverage. Device is not shown in topology.
ROUTER	Infrastructure node for extending network coverage by relaying messages. Visible in Nodes list.	Best positioned in strategic locations to maximize the network's overall coverage. Device is shown in topology.
ROUTER_CLIENT	Combination of both ROUTER and CLIENT. Not for mobile devices.	Devices in a strategic position for priority routing that need to also serve as a

Device Role	Description	Best Uses
		standard CLIENT.

Role Comparison

This table shows the **default** values after selecting a preset. As always, individual settings can be adjusted after choosing a preset.

Device Role	BLE/ WiFi/ Serial	Screen Enabled	Power Consumption	Retransmit	Prio Ro
CLIENT	Yes	Yes	Regular	Yes	No
CLIENT_MUTE	Yes	Yes	Lowest	No	No
CLIENT_HIDDEN	Yes	Yes	Lowest	Local only	No
TRACKER	Yes	No	Regular	No	No
LOST_AND_FOUND	Yes	No	Regular	No	No
SENSOR	Yes	No	High	No	No

Device Role	BLE/ WiFi/ Serial	Screen Enabled	Power Consumption	Retransmit	Prio Ro
TAK	Yes	Optional	Regular	Yes	No
TAK_TRACKER	Yes	Optional	Regular	Yes	No
ROUTER	No ¹	No	High	Yes	Yes
ROUTER_CLIENT	Yes	Yes	Highest	Yes	Yes
REPEATER	Yes	No	High	Yes	Yes

Rebroadcast Mode

This setting defines the device's behavior for how messages are rebroadcasted.

Value	Description
ALL	ALL (Default) - This setting will rebroadcast ALL messages from its primary mesh as well as other meshes with the same modem settings, including

Value	Description
	when encryption settings differ.
<code>ALL_SKIP_DECODING</code>	ALL_SKIP_DECODING - Same as behavior as ALL, but skips packet decoding and simply rebroadcasts them. Only available with Repeater role.
<code>LOCAL_ONLY</code>	LOCAL_ONLY - Ignores observed messages from foreign meshes that are open or those which it cannot decrypt. Only rebroadcasts message on the nodes local primary / secondary channels.
<code>KNOWN_ONLY</code>	KNOWN_ONLY - Ignores observed messages from foreign meshes like LOCAL_ONLY, but takes it a step further by also ignoring messages from nodenums not in the node's known list (NodeDB).

Serial Console

Acceptable values: `true` or `false`

Disabling this will disable the SerialConsole by not initializing the StreamAPI.

Debug Log

Acceptable values: `true` or `false`

By default we turn off logging as soon as an API client connects. Set this to true to leave the debug log outputting even when API is active.

GPIO for user button

This is the GPIO pin number that will be used for the user button, if your device does not come with a predefined user button.

GPIO for PWM Buzzer

This is the GPIO pin number that will be used for the PWM buzzer, if your device does not come with a predefined buzzer.

Node Info Broadcast Seconds

This is the number of seconds between NodeInfo message (containing i.a. long and short name) broadcasts from the device. The device will still respond ad-hoc to NodeInfo messages when a response is wanted. When the device hears any packet from a node it doesn't know yet, it will send its NodeInfo and ask for a

response automatically.

Double Tap as Button Press

This option will enable a double tap, when a supported accelerometer is attached to the device, to be treated as a button press.

Managed Mode

Enabling Managed mode will restrict access to all radio configurations via client applications. Radio configurations will only be accessible through the Admin channel. To avoid being locked out, make sure the Admin channel is working properly before enabling it.

Device Config Client Availability

Android

ⓘ INFO

Device Config is available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Device**

Apple

ⓘ INFO

All device config options other than NTP Server are available on iOS, iPadOS and macOS at Settings > Device Configuration > Device.

CLI

ⓘ INFO

All device config options are available in the python CLI.
Example commands are below:

Setting	Acceptable Values	Default
device.debug_log_enabled	true, false	false
device.role	CLIENT, CLIENT_MUTE, ROUTER, ROUTER_CLIENT, REPEATER, TRACKER, SENSOR	CLIENT
device.rebroadcast_mode	ALL, ALL_SKIP_DECODING, LOCAL_ONLY	ALL

Setting	Acceptable Values	Default
device.serial_enabled	true, false	true
device.button_gpio	0 - 34	0
device.buzzer_gpio	0 - 34	0
device.node_info_broadcast_secs	0 - UINT_MAX	10800 (3 hours)
device.double_tap_as_button_press	false, true	false
device.is_managed	false, true	false

TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set device.role CLIENT --set
device.debug_log_enabled true
```

Set the role to client

```
meshtastic --set device.role CLIENT
```

Disable serial console

```
meshtastic --set device.serial_enabled false
```

Enable debug logging

```
meshtastic --set device.debug_log_enabled true
```

Web



All device config options are available in the Web UI.

Footnotes

The Router role enables Power Saving by default. Consider ROUTER_CLIENT if BLE/WiFi/Serial are still needed. ↩

Display Configuration

The display config options are: Screen On Duration, Auto Carousel Interval, Always Point North, GPS Format, Preferred Display Units, OLED Definition, Display Mode, Heading Bold, and Wake on Tap or Motion. Display config uses an admin message sending a `Config.Display` protobuf.

Display Config Values

Screen On Duration

How long the screen remains on after the user button is pressed or messages are received.

Auto Carousel Interval

Automatically toggles to the next page on the screen like a carousel, based on the specified interval.

Always Point North

If this is set, the compass heading on the screen outside of the circle will always point north. This feature is off by default and the top of display represents your heading direction, the North indicator will move around the circle.

GPS Format

The format used to display GPS coordinates on the device screen.

Acceptable values:

Value	Description
DEC	Decimal Degrees
DMS	Degrees Minutes Seconds
UTM	Universal Transverse Mercator
MGRS	Military Grid Reference System
OLC	Open Location Code (Plus Codes)
OSGR	Ordnance Survey Grid Reference

Preferred Display Units

Switch between METRIC (default) and IMPERIAL units

Flip Screen

If enabled, the screen will be rotated 180 degrees, for cases that

mount the screen upside down

OLED Definition

The type of OLED Controller is auto-detected by default, but can be defined with this setting if the auto-detection fails. For the SH1107, we assume a square display with 128x128 Pixels like the GME128128-1.

Acceptable values:

Value	Description
OLED_AUTO	Auto detect display controller
OLED_SSD1306	Always use SSD1306 driver
OLED_SH1106	Always use SH1106 driver
OLED_SH1107	Always use SH1107 driver (Geometry 128x128)

Display Mode

The display mode can be set to `DEFAULT` (default), `TWOCOLOR`, `INVERTED` or `COLOR`. The `TWOCOLOR` mode is intended for OLED displays with the first line of output being a different color than the

rest of the display. The **INVERTED** mode will invert that bicolor area, resulting in a white background headline on monochrome displays.

Heading Bold

The heading can be hard to read when 'INVERTED' or 'TWOCOLOR' display mode is used. This setting will make the heading bold, so it is easier to read.

Wake on Tap or Motion

This option enables the ability to wake the device screen when motion, such as a tap on the device, is detected via an attached accelerometer.

Display Config Client Availability

Android

ⓘ INFO

Display Config is available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Display**

Apple

ⓘ INFO

All display config options are available on iOS, iPadOS and macOS at Settings > Device Configuration > Display.

CLI

ⓘ INFO

All display config options are available in the python CLI.

Example commands are below:

Setting	Acceptable Values	Default
display.auto_screen_carousel_secs	integer	Default of 0 is off.
display.compass_north_top	false, true	false
display.flip_screen	false, true	false
display.gps_format	DEC, DMS, UTM, MGRS, OLC, OSGR	DEC
display.oled	OLED_AUTO, OLED_SSD1306, OLED_SH1106,	OLED_AUTO

Setting	Acceptable Values	Default
	OLED_SH1107	
display.screen_on_secs	integer	Default of 0 is 10 minutes.
display.units	METRIC, IMPERIAL	METRIC
display.displaymode	DEFAULT, TWOCOLOR, INVERTED, COLOR	DEFAULT
display.heading_bold	false, true	false
display.wake_on_tap_or_motion	false, true	false

TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a

config section that commands be chained together as one.

Example:

```
meshtastic --set display.screen_on_secs 120 --  
set display.gps_format UTM
```

Set Screen On Duration (Default of 0 is 10 minutes)

```
meshtastic --set display.screen_on_secs 0  
meshtastic --set display.screen_on_secs 120
```

Set Auto Carousel Interval (Default of 0 is Off)

```
meshtastic --set display.auto_screen_carousel_secs 0  
// Set to 2 Minutes (120 Seconds)  
meshtastic --set display.auto_screen_carousel_secs  
120
```

Specify GPS format on device screen

```
meshtastic --set display.gps_format UTM
```

Web



All display config options are available in the Web UI.

LoRa Configuration

The LoRa config options are: Region, Modem Preset, Max Hops, Transmit Power, Bandwidth, Spread Factor, Coding Rate, Frequency Offset, Transmit Enabled, Channel Number, Ignore Incoming Array, Ignore MQTT, Override Duty Cycle Limit, SX126x RX Boosted Gain, and Override Frequency. LoRa config uses an admin message sending a `Config.LoRa` protobuf.

 **NOTE**

In order to communicate fully, devices within a mesh must have identical settings for Region and Modem Preset, or identical custom Modem settings.

LoRa Config Values

 **NOTE**

You must set your device's `lora.region` setting. This will ensure that you are operating within the legal limits for your area.

Region

Sets the region for your node. Default is `unset`. As long as this is

not set, the node screen will display a message and not transmit any packets.

Region Code	Description
UNSET	Unset
US	United States
EU_433	European Union 433MHz
EU_868	European Union 868MHz
CN	China
JP	Japan
ANZ	Australia & New Zealand
KR	Korea
TW	Taiwan
RU	Russia
IN	India

Region Code	Description
NZ_865	New Zealand 865MHz
TH	Thailand
UA_433	Ukraine 433MHz
UA_868	Ukraine 868MHz
MY_433	Malaysia 433MHz
MY_919	Malaysia 919MHz
SG_923	Singapore 923MHz
LORA_24	2.4 GHz band worldwide

ⓘ INFO

EU_433 and EU_868 have to adhere to an hourly duty cycle limitation of 10%. Your device will stop transmitting if you reach it, until it is allowed again.

Modem Preset

Default is `unset` which equates to `LONG_FAST`. Presets are pre-

defined modem settings (Bandwidth, Spread Factor, and Coding Rate) which influence both message speed and range. The default will provide a strong mixture of speed and range, for most users.

The presets are designed to provide further options for optimizing either speed (and reduced network congestion) or range, which can be useful for two real world scenarios:

A high number of devices exist in the mesh, or messages are sent very frequently. Faster speeds (and therefore lower radio time per device) can help with mesh network congestion.

Maximum range is desired, for long range scenarios where a several second delay in message receipt is acceptable (for instance, attempting to send messages from a town to a distant mountain top).

The Presets available are as follows, and follow a linear pattern of Fastest <--> Slowest, and Shortest <--> Longest range:

SHORT_FAST (Fastest, highest bandwidth, lowest airtime, shortest range)

SHORT_SLOW

MEDIUM_FAST

MEDIUM_SLOW

LONG_FAST (Default)

LONG_MODERATE

LONG_SLOW

VERY_LONG_SLOW (Slowest, lowest bandwidth, highest airtime, longest range. Not recommended for regular usage as does not form meshes well and is unreliable)

Max Hops

Maximum number of hops. This can't be greater than 7. Default is 3 which should be fine for most applications. **Really, 3 is fine.**



Transmit Power

If zero then, use default max legal continuous power (i.e. something that won't burn out the radio hardware)

In most cases you should use zero here. Units are in dBm.

Bandwidth

Certain bandwidth numbers are 'special' and will be converted by

the device firmware to the appropriate floating point value:

Special Value	Interpreted As
31	31.25 kHz
62	62.5 kHz
200	203.125 kHz
400	406.25 kHz
800	812.5 kHz
1600	1625.0 kHz

Please be aware that values < 62.5kHz may require a TCXO on some hardware devices.

Spread Factor

A number from 7 to 12. Indicates the number of chirps per symbol as 1[<<]spread_factor.

Coding Rate

The denominator of the coding rate. ie for 4/5, the value is 5. 4/8 the value is 8.

Frequency Offset

This parameter is for advanced users with advanced test equipment, we do not recommend most users use it.

A frequency offset that is added to the calculated band center frequency. Used to correct for crystal calibration errors.

Transmit Enabled

Allows you to enable and disable transmit (TX) from the LoRa radio. Useful for hot-swapping antennas and other tests.

Defaults to true

Frequency Slot

This setting controls the actual hardware frequency at which the radio transmits, represented by a frequency slot between 1 and NUM_SLOTS (the maximum for the current region and modem preset). If set to 0/UNSET, the device reverts to the older channel name hash-based algorithm for determining the frequency slot.

Ignore Incoming Array

For testing it is useful sometimes to force a node to never listen to particular other nodes (simulating radio out of range). All nodenums listed in the ignore_incoming array will have packets

they send dropped on receive (by router.cpp)

Ignore MQTT

Setting this option to 'true' means the device will ignore any messages it receives via LoRa that came via MQTT somewhere along the path towards the device. Note this only works when your device and the MQTT node are running at least firmware version 2.2.19.

Override Duty Cycle Limit

Setting this option to 'true' means the device will ignore the hourly duty cycle limit in Europe. This means that you might violate regulations if the device transmits too much. By default, this option is set to 'false,' which means the device will stop sending data when it reaches the hourly limit and will start again when it is allowed to do so.

SX126x RX Boosted Gain

This is an option specific to the SX126x chip series which allows the chip to consume a small amount of additional power to increase RX sensitivity.

Override Frequency

This parameter is for advanced users and licensed HAM radio

operators. When enabled, the channel calculation will be ignored, and the set frequency will be used instead (frequency_offset still applies). This will allow you to use out-of-band frequencies. Please respect your local laws and regulations. If you are a license HAM operator, make sure you enable HAM mode and turn off encryption.

LoRa Config Client Availability

Android

 **INFO**

LoRa Config options are available on Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > LoRa**

Apple

 **INFO**

All LoRa config options are available on iOS, iPadOS and macOS at Settings > Radio Configuration > LoRa.

CLI

 **INFO**

LoRa config commands are available in the python CLI.

Example commands are below:

Setting	Acceptable Values	Default
lora.modem_preset	LONG_FAST , LONG_SLOW , VERY_LONG_SLOW , MEDIUM_SLOW , MEDIUM_FAST , SHORT_SLOW , SHORT_FAST	LONG_FAST , LONG_MODERATE
lora.use_preset	false , true	false
lora.region	UNSET , US , EU_433 , EU_868 , CN , JP , ANZ , KR , TW , RU , IN , NZ_865 , TH , LORA_24 , UA_433 , UA_868 , MY_433 , MY_919 , SG_923	UNSET
lora.bandwidth	31 , 62 , 125 ,	250

Setting	Acceptable Values	Default
	250, 500	
lora.spread_factor	7, 8, 9, 10, 11, 12	12
lora.coding_rate	5, 6, 7, 8	8
lora.frequency_offset	0 to 1000000	0
lora.hop_limit	1, 2, 3, 4, 5, 6, 7	3
lora.tx_power	0 to 30	0
lora.tx_enabled	false, true	true
lora.channel_num	0, 1 to NUM_CHANNELS	0
lora.ignore_mqtt	false, true	false
lora.override_duty_cycle	false, true	false
lora.sx126x_rx_boosted_gain	false, true	false

Setting	Acceptable Values	Default
lora.override_frequency	Any supported frequency the LoRA radio is capable of. Please respect local rules and regulations	0



TIP
Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set lora.region US --set  
lora.modem_preset LONG_FAST
```

Set Modem Preset

```
meshtastic --set lora.modem_preset LONG_FAST  
meshtastic --set lora.modem_preset MEDIUM_FAST
```

Set Region

```
meshtastic --set lora.region US  
meshtastic --set lora.region EU_433
```

Set Hop Limit

```
meshtastic --set lora.hop_limit 2
```

Override Duty Cycle

```
meshtastic --set lora.override_duty_cycle true  
meshtastic --set lora.override_duty_cycle false
```

Web



INFO

All LoRa config options are available in the Web UI.

Network Configuration

The Network config options are: NTP Server, WiFi Enabled, WiFi SSID, WiFi PSK, Ethernet Enabled, IPv4 Networking Mode, and Static Address. Network config uses an admin message sending a `Config.Network` protobuf.

 **INFO**

Enabling WiFi will disable Bluetooth. Only one connection method will work at a time.

ESP32 devices have the ability to connect to WiFi as a client. SoftAP mode is not supported by the Meshtastic firmware.

Network Config Values

NTP Server

The NTP server used if IP networking is available.

Set to `0.pool.ntp.org` by default. (Max Length: 32)

WiFi Enabled

Enables or Disables WiFi.

Set to `false` (Disabled) by default.

WiFi SSID

This is your WiFi Network's SSID.

Empty `""` by default. (Case Sensitive, Max Length: 32)

WiFi PSK

This is your WiFi Network's password.

Empty `""` by default. (Case Sensitive, Max Length: 64)

Ethernet Enabled

Enables or Disables Ethernet.

Set to `false` (Disabled) by default.

IPv4 Networking Mode

Set to `DHCP` by default. Change to `STATIC` to use a static IP address. Applies to both Ethernet and WiFi.

IPv4 Static Address configuration

contains ip, gateway, subnet and dns server in case you want a static configuration.



TIP
The first time your device restarts after enabling WiFi or Ethernet, it will take an additional 20-30 seconds to boot. This is to generate self-signed SSL keys. The keys will be saved for future reuse.

Network Config Client Availability

Android



Network Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Network**

Apple



All Network config options are available on iOS, iPadOS and macOS at Settings > Device Configuration > Network.

CLI

ⓘ INFO

All Network config options are available in the python CLI.

Setting	Acceptable Values	Default
network.ntp_server	string	0.pool.ntp.org
network.wifi_enabled	true, false	false
network.wifi_ssid	string	""
network.wifi_psk	string	""
network.eth_enabled	true, false	false
network.address_mode	DHCP, STATIC	DHCP

💡 TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set network.wifi_enabled true --set  
network.wifi_ssid "my network" --set  
network.wifi_psk mypassword
```

Set NTP Server

```
meshtastic --set network.ntp_server "0.pool.ntp.org"
```

Enable / Disable WiFi

```
meshtastic --set network.wifi_enabled true  
meshtastic --set network.wifi_enabled false
```

Set WiFi SSID

```
meshtastic --set network.wifi_ssid mynetwork  
// With spaces  
meshtastic --set network.wifi_ssid "my network"
```

Set WiFi password

```
meshtastic --set network.wifi_psk mypassword  
// With spaces
```

Web



INFO

All Network config options are available in the Web UI.

Examples

WiFi Client

With `network.wifi_ssid` & `network.wifi_psk` populated, the device will know to connect to your network. Make sure you are in range of your WiFi and it is a 2.4GHz-only network. If you have a single Meshtastic device on your local network it's easy to connect to your device with DNS `http://meshtastic.local`. If you have multiple Meshtastic devices you will need to connect using their respective IP addresses.

Disable WiFi

To disable WiFi completely, set `network.wifi_enabled` to `false`.

Position Configuration

The position config options are: GPS Enabled, GPS Update Interval, GPS Attempt Time, Fixed Position, Smart Broadcast, Smart Broadcast Minimum Distance, Smart Broadcast Minimum Interval, Broadcast Interval, Position Packet Flags, and GPS RX/TX Pins.

Position config uses an admin message sending a

`Config.Position` protobuf.

Position data from GPS is provided by either the radio or your paired phone. Position data is not required to use Meshtastic but time calculations require at least one device on the mesh have either a GPS or internet connection for time.

Position Config Values

GPS Enabled

Acceptable values: `true` or `false`

Defaults to true. Enables GPS on the node.

GPS Update Interval

How often we should try to get GPS position (in seconds), or zero for the default of once every 2 minutes, or a very large value

(maxint) to update only once at boot.

GPS Attempt Time

How long should we try to get our position during each GPS update interval attempt? (in seconds) Or if zero, use the default of 15 minutes.

Fixed Position

Acceptable values: `true` or `false`

False by default

If set, this node is at a fixed position. The device will generate GPS updates at the regular GPS update interval, but use whatever the last lat/lon/alt it saved for the node. The lat/lon/alt can be set by an internal GPS or with the help of the mobile device's GPS.

Smart Broadcast

Acceptable values: `true` or `false`

True by default

Smart broadcast will send out your position at an increased frequency only if your location has changed enough for a position update to be useful.

Smart broadcast complements broadcast interval (doesn't override that setting) but will apply an algorithm to more frequently update your mesh network if you are in motion and then throttle it down when you are standing still. If you use this feature, it's best to leave broadcast interval at the default.

Smart broadcast will calculate an ideal position update interval based on the data rate of your selected channel configuration.

Smart Broadcast Minimum Distance

Default of is 100 meters

The minimum distance in meters traveled (since the last send) before we can send a position to the mesh if smart broadcast is enabled.

Smart Broadcast Minimum Interval

Default of is 30 seconds

The minimum number of seconds (since the last send) before we can send a position to the mesh if smart broadcast is enabled.

Broadcast Interval

Default of is 15 minutes

If smart broadcast is off we should send our position this often (but

only if it has changed significantly)

The GPS updates will be sent out every Broadcast Interval, with either the actual GPS location, or an empty location if no GPS fix was achieved.

Position Flags

Defines which options are sent in POSITION messages. Values are stored as a bit field of boolean configuration options (bitwise OR of PositionFlags).

Value	Description
UNSET	Required for compilation
ALTITUDE	Include an altitude value (if available)
ALTITUDE_MSL	Altitude value is MSL
GEOIDAL_SEPARATION	Include geoidal separation
DOP	Include the DOP value ; PDOP used by default, see below
HV DOP	If POS_DOP set, send separate HDOP / VDOP values instead of PDOP

Value	Description
SATINVIEW	Include number of "satellites in view"
SEQ_NO	Include a sequence number incremented per packet
TIMESTAMP	Include positional timestamp (from GPS solution)
HEADING	Include positional heading (from GPS solution)
SPEED	Include positional speed (from GPS solution)

GPIO RX/TX for GPS Module

If your device does not have a fixed GPS chip, you can define the GPIO pins for the RX and TX pins of a GPS module.

Position Config Client Availability

Android



INFO

Position Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Position**

Apple

! **INFO**

All position config values are available on iOS, iPadOS and macOS at Settings > Device Configuration > Position.

CLI

! **INFO**

All Position config commands are available in the python CLI.

Example commands are below:

Setting	Acceptable Values
position.gps_enabled	true, false
position.gps_update_interval	integer (seconds)
position.gps_attempt_time	integer (seconds)

Setting	Acceptable Values
position.fixed_position	true, false
position.position_broadcast_smart_enabled	true, false
position.broadcast_smart_minimum_distance	integer (meters)
position.broadcast_smart_minimum_interval_secs	integer (seconds)
position.position_broadcast_secs	integer (seconds)
position.flags	UNSET, ALTITUDE, ALTITUDE_MSL,

Setting	Acceptable Values
	GEOIDAL_SEPARATION , DOP , HVDOP , PDOP , SATINVIEW , SEQ_NO , TIMESTAMP , HEADING , SPEED
position.rx_gpio	integer (0-39)
position.tx_gpio	integer (0-34)

TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

This is especially important for position values to ensure they are set at the same time and avoid being overwritten by subsequent commands.

Example:

```
meshtastic --set position.fixed_position true --  
setlat 37.8651 --setlon -119.5383
```

Set GPS update interval (Default of 0 is 30 seconds)

```
meshtastic --set position.gps_update_interval 0  
meshtastic --set position.gps_update_interval 45
```

Set GPS attempt time (Default of 0 is 30 seconds)

```
meshtastic --set position.gps_attempt_time 0  
meshtastic --set position.gps_attempt_time 45
```

Set Fixed Position - Current Location

```
meshtastic --set position.fixed_position true
```

ⓘ NOTE

The device will continue to acquire GPS coordinates according to the `gps_update_interval`, but will use the last saved coordinates as its fixed point.

Set Fixed Position - User Defined

```
meshtastic --setlat 37.8651 --setlon -119.5383
```

Unset Fixed Position

```
meshtastic --set position.fixed_position false
```

Enable / Disable Smart position broadcast (Enabled by default)

```
meshtastic --set  
position.position_broadcast_smart_enabled true  
meshtastic --get  
position.position_broadcast_smart_enabled false
```

Set Position Broadcast Interval (Default of 0 is 15 minutes)

```
meshtastic --set position.broadcast_secs 0  
meshtastic --set position.broadcast_secs 60
```

 **NOTE**

It may take some time to see that the change has taken effect. The GPS location is updated according to the value specified on `gps_update_interval` and the mesh will be notified of the new position in relation to the `position_broadcast_secs` value.

Set / Unset Position Flags

```
meshtastic --pos-fields ALTITUDE ALTITUDE_MSL
```

Web



All position config options are available in the Web UI.



Altering/disabling the GPS functionality does not mean that you will be unable to be found. Via triangulation of your radio, location may be given up to someone if they are determined enough.

Power Configuration

ⓘ USE DEVICE CONFIG FIRST

Power settings are advanced configuration, most users should choose a role under Device Config to manage power for their device and shouldn't ever need to adjust these settings.

The power config options are: Power Saving, Shutdown after losing power, ADC Multiplier Override, Wait Bluetooth Interval, Light Sleep Interval, Minimum Wake Interval, and Device Battery INA2xx Address. Power config uses an admin message sending a `Config.Power` protobuf.

Power Config Values

Power Saving

⚠ WARNING

If enabled, modifications to settings can be made by waking the device through pressing the user button, resetting, or through the admin channel for remote administration.

When activated, this feature disables Bluetooth, Serial, WiFi, and the device's screen to conserve power. This is particularly

beneficial for devices relying on low-current power sources, like solar panels. For details on which device roles have this feature enabled by default, please check the Device Config section.

Shutdown after losing power

Automatically shut down a device after a defined time period if power is lost.

ADC Multiplier Override

! **ESP32 ONLY**

This setting only applies to ESP32-based boards, it will have no effect on nRF52/RP2040 boards.

Ratio of voltage divider for battery pin e.g. 3.20 (R1=100k, R2=220k)

Overrides the `ADC_MULTIPLIER` defined in the firmware device variant file for battery voltage calculation.

Should be set to floating point value between 2 and 6

Calibration Process (Attribution)

Install the rechargeable battery.

Charge the battery until full. Indication of this state may vary depending on device. At this point, the battery voltage should be 4.2V +-1%.

Input the "Battery Charge Percent" displayed on the screen or in your connected app into the calculator below.

If "Battery Charge Percent" (e.g., B 3.82V 60%) is not displayed on the screen, it means that the default value of "Operative Adc Multiplier" is too high. Lower the "Operative Adc Multiplier" to a smaller number (it is recommended to decrease by 0.1) until the screen displays "Battery Charge Percent". Enter the current "Operative Adc Multiplier" in use into the "Operative Adc Multiplier" field in the calculator. Also, input the "Battery Charge Percent" displayed on the screen into the calculator.

Click the "Calculate" button to compute the "Calculated New Operative Adc Multiplier", and set it as the new "Operative Adc Multiplier" for the device.

Wait Bluetooth Interval

How long to wait before turning off BLE in no Bluetooth states

0 for default of 1 minute

Light Sleep Interval

ESP32 ONLY

This setting only applies to ESP32-based boards, it will have no effect on nRF52/RP2040 boards.

In light sleep the CPU is suspended, LoRa radio is on, BLE is off and

GPS is on

⌚ for default of five minutes

Minimum Wake Interval

While in light sleep when we receive packets on the LoRa radio we will wake and handle them and stay awake in no Bluetooth mode for this interval

⌚ for default of 10 seconds

Device Battery INA2xx Address

If an INA-2XX device is auto-detected on one of the I2C buses at the specified address, it will be used as the authoritative source for reading device battery level voltage. Setting is ignored for devices with PMUs (e.g. T-beams)

CONVERT HEXADECIMAL TO DECIMAL

I2C addresses are normally represented in hexadecimal and will require conversion to decimal in order to set via Meshtastic clients. For example the I2C address of 0x40 converted to decimal is 64.

Power Config Client Availability

Android

ⓘ INFO

Power Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Power**

Apple

ⓘ INFO

Select Power config options are available on iOS, iPadOS and macOS at Settings > Device Configuration > Power.

CLI

ⓘ INFO

All Power config options are available in the python CLI.

Setting	Acceptable Values	Default
power.is_power_saving	true, false	false
power.on_battery_shutdown_after_secs	integer	Default

Setting	Acceptable Values	Default
	(seconds)	of <code>0</code> is off
power.adc_multiplier_override	<code>2-4</code> (floating point value)	Default of <code>0</code> uses firmware values
power.wait_bluetooth_secs	integer (seconds)	Default of <code>0</code> is 1 minute
power.ls_secs	integer (seconds)	Default of <code>0</code> is 5 minutes
power.min_wake_secs	integer (seconds)	Default of <code>0</code> is 10 seconds

Setting	Acceptable Values	Default
power.device_battery_ina_address	integer (I2C address as decimal)	Default of 0 is no address set

TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set power.is_power_saving true --  
set power.on_battery_shutdown_after_secs 120
```

Enable / Disable Power Saving

```
meshtastic --set power.is_power_saving true  
meshtastic --set power.is_power_saving false
```

Enable / Disable Shutdown after losing power

```
meshtastic --set power.on_battery_shutdown_after_secs  
120  
meshtastic --set power.on_battery_shutdown_after_secs  
0
```

Set Wait Bluetooth Interval (Default of 0 is 60 seconds)

```
meshtastic --set power.wait_bluetooth_secs 0  
meshtastic --set power.wait_bluetooth_secs 120
```

Set Light Sleep to default (Default of 0 is 5 minutes)

```
meshtastic --set power.ls_secs 0  
meshtastic --set power.ls_secs 120
```

Set Minimum Wake Interval (Default of 0 is 10 seconds)

```
meshtastic --set power.min_wake_secs 0  
meshtastic --set power.min_wake_secs 120
```

Web



All power config options are available in the Web UI.

User Configuration

The user config options are: Long Name, Short Name, and Is Licensed. User config uses an admin message sending a `User` protobuf.

User Config Values

Long Name

A personalized name for your device.

Auto-generated by default.

If you are a licensed HAM operator and have enabled `IsLicensed`, this should be set to your HAM operator call sign.

Short Name

A personalized short identifier for your device.

Auto-generated by default.

Is Licensed (HAM)

If you are a licensed HAM operator and have considered the privileges and restrictions of using Meshtastic with a HAM license,

enable this flag.

Disabled by default.

By enabling `IsLicensed`, you should also review the following related configurations:

User: `LongName` (Should be your Call Sign)

Channel: `PSK` (Should be Empty, removing encryption)

User Config Client Availability

Android

 **INFO**

All User config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > User**

Apple

 **INFO**

All User config options are available on iOS, iPadOS and macOS at Settings > Device Configuration > User.

CLI

 **INFO**

All User config options are available in the python CLI.

Example commands are below:

Please see instructions for Enabling HAM License



TIP
Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set-owner 'your node name' --set-owner-short 'NODE'
```

Set the LongName Value

```
meshtastic --set-owner 'your node name'
```

Set the ShortName Value

```
meshtastic --set-owner-short 'NODE'
```

Enable Ham Mode, set name to license, disable encryption

```
meshtastic --set-ham 'CALLSIGN'
```

Web



All User config options are available in the Web UI.

Module Configuration

Modules are included in the firmware and allow users to extend the functionality of their mesh or device.

Name	Description
Ambient Lighting	Adjust the brightness of NCP5623 I2C RGB LEDs
Audio	Enable Support for Codec2 Voice Comms on certain devices.
Canned Message	Set a number of predefined messages to send out directly from the device with the use of an input device like a rotary encoder.
Detection Sensor	Configure a GPIO pin to be monitored for specified high/low status and send text alerts.
External Notification	Incoming messages are able to alert you using circuits you attach to the device (LEDs, Buzzers, etc).
MQTT	Forward packets along to an MQTT server. This allows users on the local mesh to communicate with users on another mesh over the internet.

Name	Description
Neighbor Info	Send info on 0-hop neighbors to the mesh.
Paxcounter	Count the number of BLE and Wifi devices passing by a node.
Range Test	Send messages with GPS location at an interval to test the distance your devices can communicate. Requires (at least) one device set up as a sender and one as a receiver. The receiver(s) will log all incoming messages to a CSV.
Remote Hardware	Set and read a GPIO status remotely over the mesh.
Serial Module	Send messages across the mesh by sending strings over a serial port.
Store & Forward	Stores messages on a device for delivery after disconnected clients rejoin the mesh.
Telemetry	Attach sensors to the device and transmit readings on a regular interval to the mesh.
Traceroute	Track which nodes are used to hop a message to a

Name	Description
	certain destination.

Ambient Lighting Module Usage

The Ambient Lighting Module has settings for control of onboard LEDs and allows users to adjust the brightness levels and respective color levels. Initially created for the RAK14001 RGB LED module using the NCP5623 with I2C. Config options are: LED State, Current, Red Level, Green Level, and Blue Level.

In order to use this module, make sure your devices have firmware version 2.2.5 or higher.

Ambient Lighting Config Values

LED State

Sets the LED to on or Off

Current

Sets the current for the LED output. Default is 10.

Red

Sets the red LED level. Values are 0-255.

Green

Sets the green LED level. Values are 0-255.

Blue

Sets the blue LED level. Values are 0-255.

Ambient Lighting Module Client Availability

Android

ⓘ INFO

All Ambient Lighting Module config options are available for Android in app version 2.2.3 and higher.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Ambient Lighting**

Apple

ⓘ INFO

All Ambient Lighting Module config options are available on iOS, iPadOS and macOS app versions 2.2.3 and higher at Settings > Module Configuration > Ambient Lighting

CLI

ⓘ INFO

All Ambient Lighting Module config options are available in the python CLI version 2.2.3 and higher.

Example commands are below:

Set the LED to on or off

```
meshtastic --set ambient_lighting.led_state 1  
meshtastic --set ambient_lighting.led_state 0
```

Set the led current to 5

```
meshtastic --set ambient_lighting.current 5
```

Set the red LED brightness to 103

```
meshtastic --set ambient_lighting.red 103
```

Set the green brightness to 234

```
meshtastic --set ambient_lighting.green 234
```

Set the blue LED brightness to 148

```
meshtastic --set ambient_lighting.blue 148
```

Get the Ambient Lighting Module Configuration

```
meshtastic --get ambient_lighting
```

Web



INFO

All Ambient Lighting module config options are available in the Web UI.

Audio Module Configuration

The audio module config options are: Codec2 Enabled, PTT GPIO, Audio Bitrate/Codec Mode, I2S Word Select, I2S Data IN, I2S Data OUT and I2S Clock. Audio Module config uses an admin message sending a `ConfigModule.Audio` protobuf.

With this **experimental** module, you can add a digital I2S microphone and speaker to any ESP32 device that has a SX128x radio and operates on the 2.4 GHz ISM Band. The Sub-1GHz bands are not wide enough to support continuous audio packets on the mesh, even in the Short and Fast modes. Right now, the only devices supported are the LilyGo TLora 2.1-1.8 and TLora T3S3 boards.

Audio Module Config Values

Codec2 Enabled

Enables the audio module.

PTT GPIO

The GPIO to use for the Push-To-Talk button. The default is GPIO 39

on the ESP32.

Audio Bitrate/Codec Mode

The bitrate to use for audio. The default is `CODEC2_700B`. The available options are:

- `CODEC2_DEFAULT`
- `CODEC2_3200`
- `CODEC2_2400`
- `CODEC2_1600`
- `CODEC2_1400`
- `CODEC2_1300`
- `CODEC2_1200`
- `CODEC2_700B`
- `CODEC2_700`

I2S Word Select

The GPIO to use for the WS signal in the I2S interface.

I2S Data IN

The GPIO to use for the SD signal in the I2S interface.

I2S Data OUT

The GPIO to use for the DIN signal in the I2S interface.

I2S Clock

The GPIO to use for the SCK signal in the I2S interface.

ⓘ WHAT IS THIS?

These Pins comprise an I2S digital audio interface. Meshtastic uses it in monoaural mode. The software will use the logical 'LEFT' Stereo channel for the microphone and the logical 'RIGHT' Stereo channel for the speaker, so configure your breakouts accordingly. Audio is Half-Duplex, so we can re-use part of the pins for a bi-directional configuration. There's **no** default pin assignment, setting these is mandatory.

Audio Module Config Client Availability

Android

ⓘ INFO

Audio Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Audio**

Apple

ⓘ INFO

Audio module config is not available on iOS, iPadOS and macOS.

CLI

ⓘ INFO

All audio module config options are available in the python CLI. Example commands are below:

Setting	Acceptable Values	Default
audio.codec2_enabled	true, false	false
audio.ptt_pin	GPIO Pin Number 1-39	Default of 39 is Unset
audio.bitrate	CODEC2_DEFAULT CODEC2_3200 CODEC2_2400 CODEC2_1600 CODEC2_1400 CODEC2_1300 CODEC2_1200 CODEC2_700B	CODEC2_DEFAULT

Setting	Acceptable Values	Default
	CODEC2_700	
audio.i2s_ws	GPIO Pin Number 1-34	no Default
audio.i2s_sd	GPIO Pin Number 1-39	no Default
audio.i2s_din	GPIO Pin Number 1-34	no Default
audio.i2s_sck	GPIO Pin Number 1-34	no Default



TIP
Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set audio.codec2_enabled true --set
```

Enable / Disable Module

```
meshtastic --set audio.codec2_enabled true  
meshtastic --set audio.codec2_enabled false
```

Set WS to GPIO pin number 7

```
meshtastic --set audio.i2s_ws 7
```

Set DIN to GPIO pin number 28

```
meshtastic --set audio.i2s_din 28
```

Set PTT to GPIO pin number 37

```
meshtastic --set audio.ptt_pin 37
```

Set Codec Bitrate

```
meshtastic --set audio.bitrate CODEC2_DEFAULT  
meshtastic --set audio.bitrate CODEC2_1400
```

Web



All audio module config options are available in the Web UI.

 **WARNING**

GPIO access is fundamentally dangerous because invalid options can physically damage or destroy your hardware. Ensure that you fully understand the schematic for your particular device before trying this as we do not offer a warranty. Use at your own risk.

This module requires attaching a peripheral accessory to your device. It will not work without one.

Canned Message Module Configuration

The Canned Message Module will allow you to send messages to the mesh network from the device without using the phone app. You can predefine text messages to choose from.

The canned message module config options are: Enabled, Send Bell, Messages, Input Source, Rotary Encoder Enabled, Up Down Encoder Enabled, Input Broker Pin A, Input Broker Pin B, Input Broker Pin Press, Input Broker Event Clockwise, Input Broker Event Counter Clockwise, and Input Broker Event Press. Canned Message config uses an admin message sending a `ConfigModule.CannedMessage` protobuf.

Canned Message Module Config Values

Enabled

Enables the canned message module.

Send Bell

Sends a bell character with each message.

The External Notification Module can be set up to beep when a new message arrives. This module can also be configured to beep only when a message contains the bell character.

Messages

The list of pre-set messages as configured by the user. Messages should be separated by pipes |. The total byte count for the message list can be up to 200 bytes

Input Source

Input event sources accepted by the canned message module.

Value	Description
<code>_any</code>	Default. Allows any peripheral input device connected to the device.
<code>rotEnc1</code>	Basic Rotary Encoder
<code>upDownEnc1</code>	Up Down Encoder (use this also for RAK14006 Rotary Encoder)
<code>cardkb</code>	M5 Stack CardKB (this covers RAK14004 Keymatrix)

Rotary Encoder Enabled

Enable the default rotary encoder.

Up Down Encoder Enabled

Enable the up / down encoder.

Input Broker Pin A

GPIO Pin Value (1-39) For encoder port A

Input Broker Pin B

GPIO Pin Value (1-39) For encoder port B

Input Broker Pin Press

GPIO Pin Value (1-39) For encoder Press port

Input Broker Event Clockwise

Generate the rotary clockwise event.

Input Broker Event Counter Clockwise

Generate the rotary counter clockwise event.

Input Broker Event Press

Generate input event on Press of this kind.

Canned Message Module Config Client Availability

Android



INFO

Canned Message Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Canned Message**

Apple

ⓘ INFO

All canned message module config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > Canned Messages.

CLI

ⓘ INFO

All canned message module config options are available in the python CLI.

Example commands are below:

Setting	Acceptable Values	Default
canned_message.enabled	true, false	false
canned_message.send_bell	true, false	false
canned_message.allow_input_source	rotEnc1, _any, upDownEnc1, cardkb	_any

Setting	Acceptable Values	Default
--set-canned-message	string	 (separate using pipes)
canned_message.inputbroker_event_cw	InputEventChar	(not defined)
canned_message.inputbroker_event_ccw	InputEventChar	(not defined)
canned_message.inputbroker_event_press	InputEventChar	(not defined)
canned_message.inputbroker_pin_a	integer	(not defined)
canned_message.inputbroker_pin_b	integer	(not defined)
canned_message.inputbroker_pin_press	integer	(not defined)



TIP
Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set canned_message.enabled true --  
set canned_message.send_bell true
```

Enable/Disable the Canned Message Module

```
meshtastic --set canned_message.enabled true  
meshtastic --set canned_message.enabled false
```

Enable/Disable send bell character

```
meshtastic --set canned_message.send_bell true  
meshtastic --set canned_message.send_bell false
```

Set Messages

```
meshtastic --set-canned-message "I need an  
alpinist!|Call Me|Roger Roger|Keep Calm|On my way"
```

Set Input Source

```
meshtastic --set canned_message.allow_input_source  
"_any"  
meshtastic --set canned_message.allow_input_source  
"rotEnc1"
```

Enable/Disable rotary1

```
meshtastic --set canned_message.rotary1_enabled 1
```

Set/Unset Encoder Pin A

```
meshtastic --set canned_message.inputbroker_pin_a 17  
meshtastic --set canned_message.inputbroker_pin_a 0
```

Set/Unset Encoder Pin B

```
meshtastic --set canned_message.inputbroker_pin_b 39  
meshtastic --set canned_message.inputbroker_pin_b 0
```

Set/Unset Encoder Pin Press

```
meshtastic --set canned_message.inputbroker_pin_press  
21
```

Set/Unset Input Broker CW Event

```
meshtastic --set canned_message.inputbroker_event_cw  
KEY_UP  
meshtastic --set canned_message.inputbroker_event_cw  
""
```

Set/Unset Input Broker CCW Event

```
meshtastic --set canned_message.inputbroker_event_ccw  
KEY_DOWN  
meshtastic --set canned_message.inputbroker_event_ccw  
""
```

Set/Unset Input Broker Press Event

```
meshtastic --set  
canned_message.inputbroker_event_press KEY_SELECT  
meshtastic --set  
canned_message.inputbroker_event_press ""
```

Web



All canned message module config options are available in the Web UI.

⚠️ WARNING

GPIO access is fundamentally dangerous because invalid options can physically damage or destroy your hardware. Ensure that you fully understand the schematic for your particular device before trying this as we do not offer a warranty. Use at your own risk.

This module requires attaching a peripheral accessory to your device. It will not work without one.

Hardware

To navigate through messages and select one, you will require some hardware attached to your device. Currently, the module is tested with a generic rotary encoder, an up/down/select 3 button logic and several I2C Keyboards. Further input methods will be added in the future.

I2C Keymatrix

This is tested with the RAK14004 Keyboard. A keypress will immediately send the message attached to the button number. Buttons are numbered from upper left to lower right on the keypanels. So pressing the upper left button will send the first message. The second button will send the second message and so on.

Caveat: the RAK 3x4 keymatrix is missing the 4th button row while scanning, so you have to skip every 4th message slot. Button 1 sends message 1 and button 4 will send message 5
Example: 1|2|3||5|6|7||9|10|11||13|14|15 - the slots 4,8 and 12 can not be used.

CardKB

The CardKB is fully supported in freetext mode and select mode. Use UP/DOWN/ENTER to select a predefined message and send it. For a freetext message, just type it in and press ENTER to send it.

If you don't want to broadcast your freetext message, you can use the CardKB to send it to a specific node. Just press TAB and select the target node with the LEFT/RIGHT keys. The message will be sent to the node with the matching name and node number. The target node will be remembered for your next message.

3 Button up/down and RAK rotary encoder

Just use UP/DOWN/ENTER to select a predefined message and send it.

Rotary encoder

Meshtastic supports hardwired rotary encoders as input devices.

You will need a generic rotary encoder. The types listed below has

five legs where two is dedicated to a "press" action, but any other types will likely do the job. You can also use a three-legged version, where the "press" action should be wired from an independent switch.

[Amazon link](#)

[Amazon.DE link](#)

[Aliexpress link1](#)

[Aliexpress link2](#)

[Aliexpress link3](#)

Connect your rotary encoder as follows. The rotary encoder has two rows of legs. One of the rows contains two legs, the other contains three legs. Bottom side view:

```
B o --- o PRESS  
GND o | |  
A o --- o GND
```

The two legs is to sense the press action (or push). Connect one of the two to GROUND and the other to a GPIO pin. (No matter which one goes where.) Let's call this connected ports 'PRESS'.

The three legs is to sense the rotation action. Connect the middle leg to GROUND and the ones on the side to GPIO pins. Let's call these ports 'A' and 'B', according to the scheme below.

```
A  --|||  
GND --|||=-----  
B  --|||
```

Recommended GPIO pins for connecting a rotary encoder.

TTGO LoRa V1:

A - GPIO-22

B - GPIO-23

PRESS - GPIO-21

There is a reference case 3D-design utilizing the rotary encoder for TTGO LoRa V1: Case for TTGO-ESP32-LORA-OLED-v1.0 with rotary encoder

Examples

Attach a compatible peripheral device. Take note of the GPIO numbers you use, as they will be used in the following step.

 **NOTE**

Replace each `GPIO` (x3) below with the GPIO numbers from your hardware setup.

```
Canned Message Module - Required Rotary Encoder Module  
Settings
```

```
meshtastic --set canned_message.inputbroker_pin_a  
GPIO  
meshtastic --set canned_message.inputbroker_pin_b  
GPIO  
meshtastic --set canned_message.inputbroker_pin_press  
GPIO  
meshtastic --set canned_message.inputbroker_event_cw  
KEY_UP  
meshtastic --set canned_message.inputbroker_event_ccw  
KEY_DOWN  
meshtastic --set  
canned_message.inputbroker_event_press KEY_SELECT  
meshtastic --set canned_message.rotary1_enabled True
```

That's it! With a functioning and enabled rotary encoder, you're ready to begin configuring the Canned Message Module.

Detection Sensor Module Usage

The Detection Sensor module allows you to configure a GPIO pin to be monitored for a specified high/low status and send text alerts over the Detection Sensor portnum when an event is detected. This is particularly useful for motion detection sensors, reed switches, and other open / closed state systems in which notifications over the mesh are desired. Config options are: Enabled, Minimum Broadcast Interval, State Broadcast Interval, Send Bell, Name, Monitor Pin, Detection Triggered High, and Use Pull-up.

In order to use this module, make sure your devices have firmware version 2.2.2 or higher.

Detection Sensor Module Config Values

Enabled

Whether the Module is enabled.

Minimum Broadcast Interval

The interval in seconds of how often we can send a message to the mesh when a state change is detected.

State Broadcast Interval

The interval in seconds of how often we should send a message to the mesh with the current state regardless of changes, When set to 0, only state changes will be broadcasted, Works as a sort of status heartbeat for peace of mind.

Send Bell

Send ASCII bell with alert message. Useful for triggering ext. notification on bell name.

Friendly Name

Used to format the message sent to mesh. Example: A name "Motion" would result in a message "Motion detected". Maximum length of 20 characters.

Monitor Pin

The GPIO pin to monitor for state changes.

Detection Triggered High

Whether or not the GPIO pin state detection is triggered on HIGH (1), otherwise LOW (0).

Use Pull-up

Whether or not use INPUT_PULLUP mode for GPIO pin. Only applicable if the board uses pull-up resistors on the pin.

Detection Sensor Module Client Availability

Android

ⓘ INFO

All Detection Sensor Module config options are available for Android in app version 2.2.2 and higher.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Detection Sensor**

Apple

ⓘ INFO

All Detection Sensor Module config options are available on iOS, iPadOS and macOS app versions 2.2.2 and higher at Settings > Module Configuration > Detection Sensor

CLI



INFO

All Detection Sensor Module config options are available in the python CLI version 2.2.2 and higher.

Example commands are below:

Enable/Disable the Detection Sensor Module

```
meshtastic --set detection_sensor.enabled true  
meshtastic --set detection_sensor.enabled false
```

Set the Minimum Broadcast Interval to 90 seconds

```
meshtastic --set  
detection_sensor.minimum_broadcast_secs 90
```

Set the State Broadcast Interval to 5 minutes

```
meshtastic --set  
detection_sensor.state_broadcast_secs 300
```

Enable/Disable Send Bell

```
meshtastic --set detection_sensor.send_bell true  
meshtastic --set detection_sensor.send_bell false
```

Set the friendly name to 'motion'

```
meshtastic --set detection_sensor.name "motion"
```

Set the Monitor Pin to 7

```
meshtastic --set detection_sensor.monitor_pin 7
```

Enable Notifications when the Monitor Pin goes HIGH

```
meshtastic --set  
detection_sensor.detection_triggered_high true
```

Enable Notifications when the Monitor Pin goes LOW

```
meshtastic --set  
detection_sensor.detection_triggered_high false
```

Enable the use INPUT_PULLUP mode

```
meshtastic --set detection_sensor.use_pullup true
```

Get the Detection Sensor Module Configuration

```
meshtastic --get detection_sensor
```

Web



All Detection Sensor module config options are available in the Web UI.

External Notification Module Configuration

The External Notification Module will allow you to connect a buzzer, speaker, LED, or other device to notify you when a message has been received from the mesh network. You can enable up to 3 pins independently from each other.

The External Notification Module config options are: Enabled, Active, Alert Bell (General), Alert Bell Vibra, Alert Bell Buzzer, Alert Message (General), Alert Message Vibra, Alert Message Buzzer, Output (General), Output Vibra, Output Buzzer, Output Milliseconds, Use PWM, and Nag Timeout. External Notification config uses an admin message sending a

`ConfigModule.ExternalNotificationConfig` protobuf.

External Notification Module Config Values

Enabled

Enables the external notification module.

Active (general / LED only)

Specifies whether the external circuit is active when the device's GPIO is low or high. If this is set true, the pin will be pulled active high, false means active low.

Alert when receiving a bell (general / LED, Vibra and Buzzer)

Specifies if an alert should be triggered when receiving an incoming bell.

Alert when receiving a message (general / LED, Vibra and Buzzer)

Specifies if an alert should be triggered when receiving an incoming message.

GPIO Pins (general / LED, Vibra and Buzzer)

Specifies the GPIO that your external circuit is attached to on the device. On devices that have a PWM buzzer, you can use the buzzer for notifications by setting the `use_pwm` property to TRUE. The Buzzer Pin will be ignored and the `device.buzzer_gpio` is used instead. If you enable PWM mode, the device will use so-called RTTTL ring tones for notification. You can find examples of RTTTL ring tones here and upload them to the device via a client application.

 **INFO**

On ESP32 based boards, GPIOs 34 to 39 are GPIOs – input only pins. These pins do not have internal pull-up or pull-down resistors. They can not be used as outputs, so you can NOT use these pins as outputs.

How long monitored GPIO is triggered

Specifies how long in milliseconds you would like your GPIOs to be active. In case of the repeat option, this is the duration of every tone and pause.

Default of 0 is 1000ms

Repeat (Nag Timeout) (general / LED, Vibra and Buzzer)

Specifies if the alert should be repeated. If set to a value greater than zero, the alert will be repeated until the user button is pressed or 'value' number of seconds have past.

External Notification Module Config Client Availability

Android

 **INFO**

External Notification Config options are available for Android.
Open the Meshtastic App
Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > External Notification**

Apple

 **INFO**

All external notification module config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > External Notification.

CLI

 **INFO**

All external notification module config options are available in

the python CLI. Example commands are below:

Setting	Acceptable Values	Default
external_notification.enabled	true, false	false
external_notification.active	true, false	false
external_notification.alert_bell	true, false	false
external_notification.alert_bell_vibra	true, false	false
external_notification.alert_bell_buzzer	true, false	false
external_notification.alert_message	true, false	false
external_notification.alert_message_vibra	true, false	false
external_notification.alert_message_buzzer	true, false	false
external_notification.output	integer	0
external_notification.output_vibra	integer	0

Setting	Acceptable Values	Default
external_notification.output_buzzer	integer	0
external_notification.output_ms	integer (milliseconds)	0
external_notification.use_pwm	true, false	false
external_notification.nag_timeout	integer (seconds)	0

 **TIP**

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set external_notification.enabled
true --set external_notification.alert_bell true
```

Enable/Disable External Notification Module

```
meshtastic --set external_notification.enabled true  
meshtastic --set external_notification.enabled false
```

Enable/Disable alert on incoming bell

```
meshtastic --set external_notification.alert_bell  
true  
meshtastic --set external_notification.alert_bell  
false
```

Set GPIO active high / low (default of false is low)

```
meshtastic --set external_notification.active false  
meshtastic --set external_notification.active true
```

Enable/Disable alert on incoming message

```
meshtastic --set external_notification.alert_message  
true  
meshtastic --set external_notification.alert_message  
false
```

Set GPIO to monitor to 21

```
meshtastic --set external_notification.output 21
```

Set monitored GPIO output duration (default of 0 is 1000ms)

```
meshtastic --set external_notification.output_ms 0  
meshtastic --set external_notification.output_ms 1500
```

Web



INFO

All External Notification module config is available for the Web UI.



WARNING

GPIO access is fundamentally dangerous because invalid options can physically damage or destroy your hardware. Ensure that you fully understand the schematic for your particular device before trying this as we do not offer a warranty. Use at your own risk.

This module requires attaching a peripheral accessory to your device. It will not work without one.

Examples

Alert Types

We support being alerted on two events:

Incoming Text Message

Incoming Text Message that contains the ASCII bell character. At present, only the Python API can send an ASCII bell character, but more support may be added in the future.

 **INFO**

The bell character is ASCII 0x07. Include 0x07 anywhere in the text message and with `ext_notification.alert_bell` enabled, we will issue an external notification.

External Hardware

Be mindful of the max current sink and source of the ESP32 GPIO. The easiest devices to interface with would be either an LED or Active Buzzer.

Ideas for external hardware:

LED

Active Buzzer

Flame thrower

Strobe Light

Siren

Known Limitations

This module only monitors text messages. We won't trigger on any other packet types.

MQTT Module Configuration

If your device is connected to Internet via wifi or ethernet, you can enable it to forward packets along to an MQTT server. This allows users on the local mesh to communicate with users on the internet. One or more channels must also be enabled as uplink and/or downlink for packets to be transmitted from and/or to your mesh (See channels). Without these settings enabled, the node will still connect to the MQTT server but only send status messages.

The MQTT module config options are: Enabled, Server Address, Username, Password, Encryption Enabled, JSON Enabled, TLS Enabled, and Root Topic. MQTT Module config uses an admin message sending a `ConfigModule.MQTT` protobuf.

Settings

MQTT Module Config Values

Enabled

Enables the MQTT module.

Server Address

The server to use for MQTT. If not set, the default public server will be used.

Username

MQTT Server username to use (most useful for a custom MQTT server). If using a custom server, this will be honored even if empty. If using the default public server, this will only be honored if set, otherwise the device will use the default username.

Password

MQTT password to use (most useful for a custom MQTT server). If using a custom server, this will be honored even if empty. If using the default server, this will only be honored if set, otherwise the device will use the default password.

Encryption Enabled

Whether to send encrypted or unencrypted packets to MQTT. This parameter is only honored if you also set server (the default official mqtt.meshtastic.org server can handle encrypted packets). Unencrypted packets may be useful for external systems that want to consume meshtastic packets.

JSON Enabled

 **NOTE**

JSON is not supported on the nRF52 platform.

Enable the sending / consumption of JSON packets on MQTT. These packets are not encrypted, but offer an easy way to integrate with systems that can read JSON.

TLS Enabled

If true, we attempt to establish a secure connection using TLS.

Root Topic

The root topic to use for MQTT messages. This is useful if you want to use a single MQTT server for multiple meshtastic networks and separate them via ACLs.

MQTT Module Config Client Availability

Android

 **INFO**

MQTT Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > MQTT**

Apple

 **INFO**

All MQTT config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > MQTT.

CLI

 **INFO**

All MQTT module config options are available in the python CLI. Example commands are below:

Setting	Acceptable Values	Default
mqtt.enabled	true, false	false
mqtt.address	string	mqtt.meshtastic.org
mqtt.username	string	meshdev
mqtt.password	string	large4cats

Setting	Acceptable Values	Default
mqtt.encryption_enabled	true, false	false
mqtt.json_enabled	true, false	false
mqtt.tls_enabled	true, false	false
mqtt.root	string	

 **TIP**

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set mqtt.enabled true --set
mqtt.json_enabled true
```

Enable/Disable MQTT Module

```
meshtastic --set mqtt.enabled true  
meshtastic --set mqtt.enabled false
```

Enable/Disable MQTT JSON

```
meshtastic --set mqtt.json_enabled true  
meshtastic --set mqtt.json_enabled false
```

Web



All MQTT module config options are available for the Web UI.

Connect to the Default Public Server

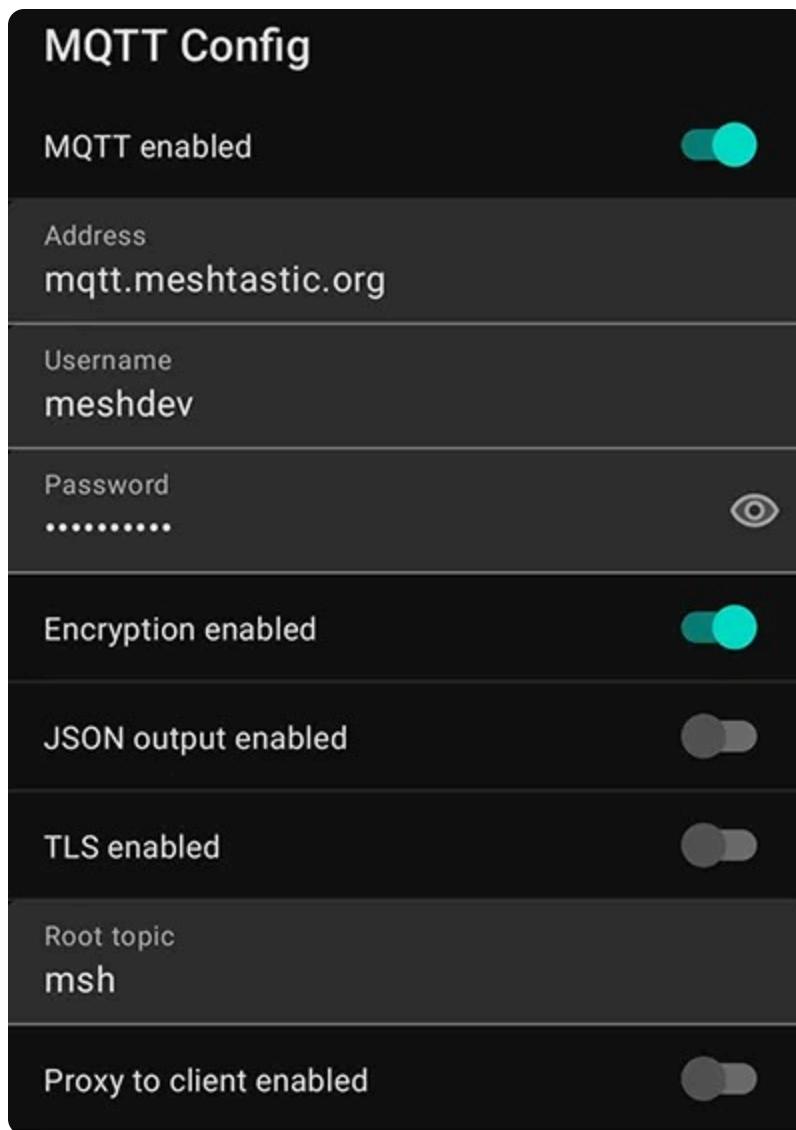


The default channel (LongFast) on the public server usually has a lot of traffic. Your device may get overloaded and may no longer function properly anymore. It is recommended to use a different channel or to use your own MQTT server if you experience issues.

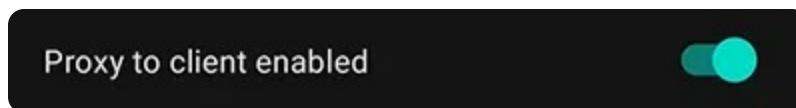
Android

1. Enable the MQTT Module

Navigate to: Vertical Ellipsis (3 dots top right) > Radio configuration > MQTT: Turn on the slider for **MQTT enabled** and tap **Send**.

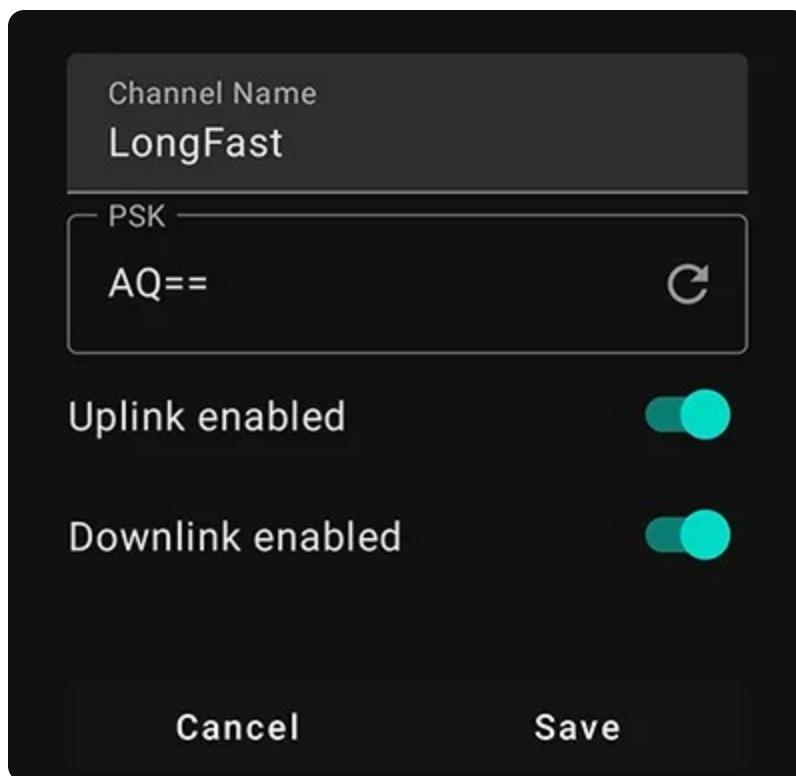


Optional: To use your phone's internet connection to send and receive packets over the web, also enable the slider for **MQTT Client Proxy** and skip the Configure Network Settings step below.



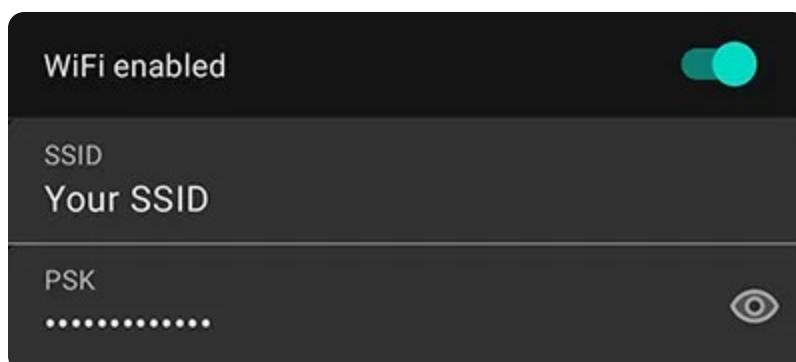
2. Enable Channel Uplink & Downlink

Navigate to: Vertical Ellipsis (3 dots top right) > Radio configuration > Channels > LongFast: Turn on the sliders for **Uplink enabled** and **Downlink enabled**, then tap **Save** and tap **Send**.



3. Configure Network Settings

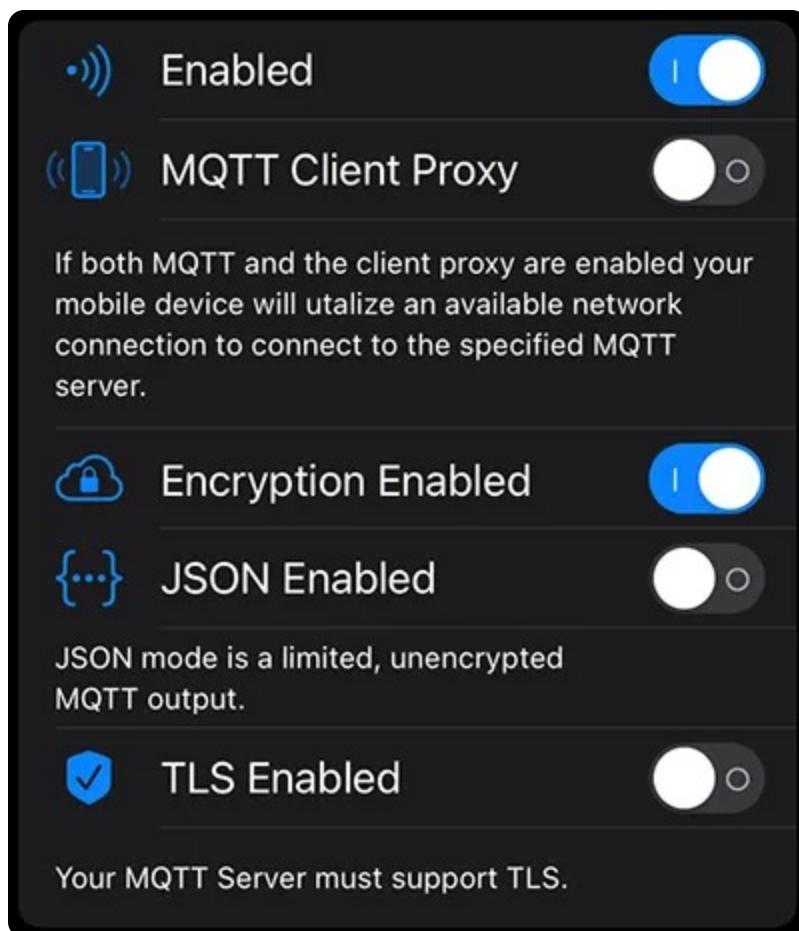
Navigate to: Vertical Ellipsis (3 dots top right) > Radio configuration > Network: Turn on the slider for **WiFi enabled**, Enter the **SSID** and **PSK** for your network, then tap **Send**.



Apple

1. Enable the MQTT Module

Navigate to Settings > MQTT: Turn on the slider for MQTT enabled and tap **Save**



 Address mqtt.meshastic.org

 Username meshdev

 Password large4cats

 Root Topic Root Topic

The root topic to use for MQTT messages. Default is "msh". This is useful if you want to use a single MQTT server for multiple meshastic networks and separate them via ACLs

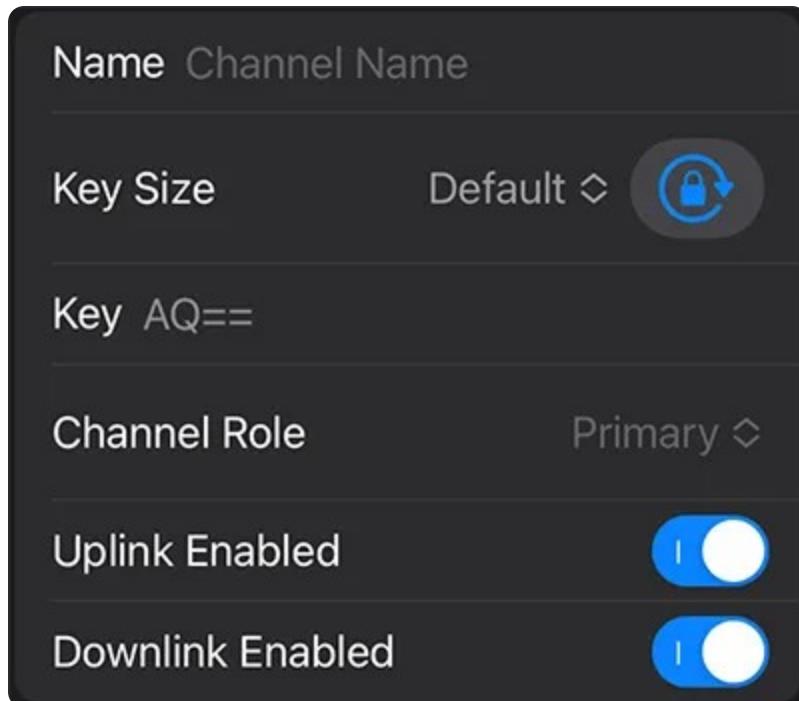
Optional: To use your phone's internet connection to send and receive packets over the web, also enable the slider for **MQTT Client Proxy** and skip the Configure Network Settings step below.

 MQTT Client Proxy



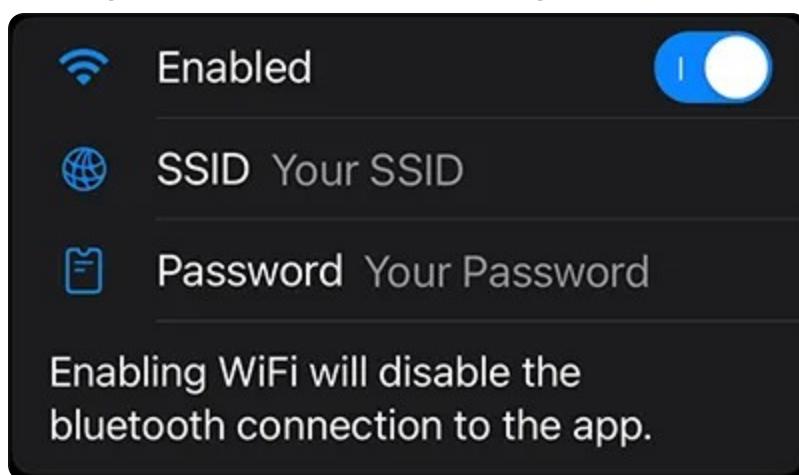
2. Enable Channel Uplink & Downlink

Navigate to Settings > Channels > Primary Channel: Turn on the sliders for **Uplink enabled** and **Downlink enabled** - Tap **Save**



3. Configure Network Settings

Navigate to Settings > Network: Turn on the slider for **WiFi enabled** - Enter your **SSID** and **PSK** for your network - Tap **Save**



CLI

1. Enable the MQTT Module

```
meshtastic --set mqtt.enabled true
```

2. Enable Channel Uplink & Downlink

```
meshtastic --ch-set uplink_enabled true --ch-index 0  
meshtastic --ch-set downlink_enabled true --ch-index  
0
```

or chained together:

```
meshtastic --ch-set uplink_enabled true --ch-index 0  
--ch-set downlink_enabled true --ch-index 0
```

3. Configure Network Settings

```
meshtastic --set network.wifi_enabled true  
meshtastic --set network.wifi_ssid "your network"  
meshtastic --set network.wifi_psk yourpassword
```

or chained together:

```
meshtastic --set network.wifi_enabled true --set  
network.wifi_ssid "your network" --set  
network.wifi_psk yourpassword
```

Web

1. Enable the MQTT Module

Navigate to Config > Module Config > MQTT - Turn on the slider for MQTT enabled - Click the **Save** icon.

MQTT Settings
Settings for the MQTT module

Enabled	Enable or disable MQTT
<input checked="" type="checkbox"/>	
MQTT Server Address	MQTT server address to use for default/custom servers
mqtt.meshastic.org	
MQTT Username	MQTT username to use for default/custom servers
meshdev	
MQTT Password	MQTT password to use for default/custom servers

Encryption Enabled	Enable or disable MQTT encryption
<input checked="" type="checkbox"/>	
JSON Enabled	Whether to send/consume JSON packets on MQTT
<input checked="" type="checkbox"/>	
TLS Enabled	Enable or disable TLS
<input checked="" type="checkbox"/>	
Root topic	MQTT root topic to use for default/custom servers
msh	
Proxy to Client Enabled	Whether to proxy MQTT packets to the client (for example to Home Assistant)
<input checked="" type="checkbox"/>	

Optional: To use your client's internet connection to send and receive packets over the web, also enable the slider for **Proxy to Client Enabled** and skip the Configure Network Settings step below.

Proxy to Client Enabled	Whether to proxy MQTT packets to the client (for example to Home Assistant)
<input checked="" type="checkbox"/>	

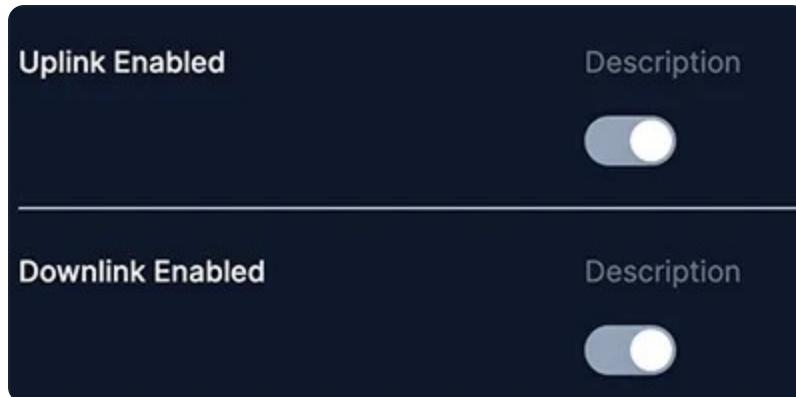
⚠ CAUTION

Though this option may be visible in your UI, Client Proxy is

not yet functional with the Web Client.

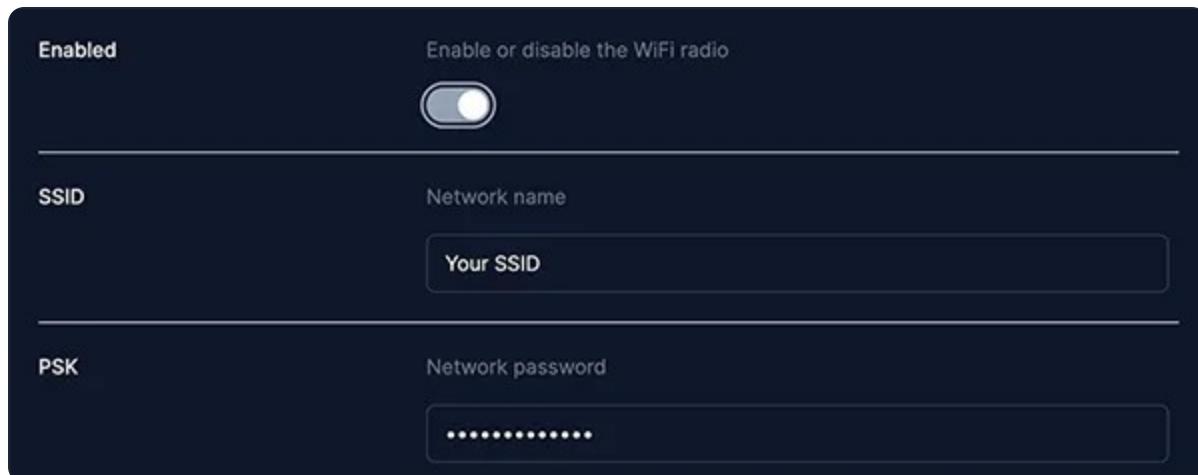
2. Enable Channel Uplink & Downlink

Navigate to Channels > Primary: Turn on the sliders for **Uplink Enabled** and **Downlink Enabled** - Click the **Save** icon.



3. Configure Network Settings

Navigate to Radio Config > Device > Network: Turn on the slider for **Enabled** - Enter your **SSID** and **PSK** for your network - Click the **Save** icon.



Neighbor Info Module Usage

The Neighbor Info Module is for sending information on each node's 0-hop neighbors to the mesh. Config options are: Enabled and Update Interval.

In order to save bandwidth, Meshtastic does not keep track of the route a packet is taking to get to the destination, except when using Traceroute. When enabling this module, your node will periodically send out a packet with a list of its direct neighbors and the quality (SNR) of the corresponding link. If you enable this on all nodes, you can build up a graph of the full network to see how it is connected. At the time of writing, there are no clients visualizing it for you, but you can use e.g. MQTT to gather all the data.

In order to use it, make sure your devices use firmware version 2.2.0 or higher.

Neighbor Info Module Config Values

Enabled

Enables the Neighbor Info Module.

Update Interval

How often the neighbor info is sent to the mesh.

Neighbor Info Module Client Availability

Android



All Neighbor Info Module config options are available for Android in app version 2.2.0 and higher.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Neighbor Info**

Apple

Not yet implemented.



All Neighbor Info Module config options are available in the python CLI version 2.2.0 and higher.

Example commands are below:

Enable/Disable the Neighbor Info Module

```
meshtastic --set neighbor_info.enabled true
```

Set the update interval to 10 minutes

```
meshtastic --set neighbor_info.update_interval 600
```

Get the Neighbor Info Module Configuration

```
meshtastic --get neighbor_info
```



All Neighbor Info module config options are available in the Web UI.

Paxcounter Module Usage

The Paxcounter module counts the number of people passing by a specific area by scanning for WiFi and BLE MAC addresses. It is commonly used in retail stores, museums, and other public spaces to monitor foot traffic and gather valuable data for analysis.

In order to use this module, make sure your devices have firmware version 2.2.17 or higher.

 **INFO**

This module can only be used with ESP32 devices. To operate the Paxcounter Module, it is mandatory to switch off both WiFi and Bluetooth in your Network and Bluetooth settings.

Paxcounter Module Config Values

Enabled

Whether the Module is enabled.

Update Interval

The interval in seconds of how often we can send a message to

the mesh when a state change is detected.

Paxcounter Module Client Availability

Android

 **INFO**

Paxcounter Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Paxcounter**

Apple

 **INFO**

All Paxcounter config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > Paxcounter.

CLI

 **INFO**

All Paxcounter Module config options are available in the python CLI version 2.2.16 and higher.

Example commands are below:

Enable/Disable the Paxcounter Module

```
meshtastic --set paxcounter.enabled true
```

Set the Minimum Broadcast Interval to 900 seconds

```
meshtastic --set  
paxcounter.paxcounter_update_interval 900
```

Get the Paxcounter Module Configuration

```
meshtastic --get paxcounter
```

Web



All Paxcounter module config options are available in the Web UI.

Range Test Module Configuration

This module allows you to test the range of your Meshtastic nodes. It requires two nodes:

Sender: fixed node that sends sequential packets ("Sender message interval" between 30-60s);

Receiver: mobile node (typically you) with onboard or phone GPS.

Nodes are in range as long as the sequential packets can be received.

The receiving node can be used to save the messages along with the GPS coordinates at which they were received into a .csv file.

This .csv file can then be integrated into Google Earth, Google Maps - My Maps, or any other program capable of processing .csv files. This can enable you to visualize your mesh.

 **INFO**

Be sure to turn off the module or disable sending when not in use. This will use a lot of time on air, slow down your mesh, and spam your channel. The module will automatically turn off after 8 hours.

The range test module config options are: Enabled, Sender, and Save. Range Test Module config uses an admin message sending a `ModuleConfig.RangeTestConfig` protobuf.

Range Test Module Config Values

Enabled

Enables the range test module. **Both Sender and Receiver must have the module enabled.**

Sender Interval

How long to wait between sending sequential test packets. 0 is default which disables sending messages.

Recommended Sender Settings

Radio Setting	<code>range_test.sender</code>
Long Slow	60
Long Fast	30
Medium	15

Radio Setting	range_test.sender
Short Fast	15

Save CSV File



INFO

Leave disabled when using the Android or Apple apps.

Saves directly to the device's flash memory (without the need for a smartphone), and is only available on ESP32-based devices.

If enabled, all received messages are saved to the device's flash memory in a file named rangetest.csv.

To access this file, first turn on the WiFi on your device and connect to your network. Once you can connect to your device, navigate to meshtastic.local/rangetest.csv (or your_device_ip/rangetest.csv) and the file will be downloaded automatically. This file will only be created after receiving initial messages.

To prevent filling up the storage, the device will abort writing if there is less than 50kb of space on the filesystem.

Range Test Module Config Client Availability

Android



Range Test Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Range Test**

Android exports a rangetest.csv file from packets in the Debug Log. To clear old packet history data: [Debug Panel > Clear](#)

Apple



All range test module config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > Range Test.

Apple apps also have the option to download logged position data which is stored on your iPhone/iPad/Mac. Access this by clicking on the Nodes tab, selecting a node, then select Position Log and click Save. This data file does not require the Range Test module to be active.

CLI

ⓘ INFO

Range Test module config options are available in the python CLI. Example commands are below:

Setting	Acceptable Values	Default
range_test.enabled	true, false	false
range_test.save	true, false	false
range_test.sender	integer (Seconds)	0

💡 TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set range_test.enabled true --set  
range_test.save false
```

Enable / Disable the range test

```
meshtastic --set range_test.enabled true  
meshtastic --set range_test.enabled false
```

Enable / Disable range test save

```
meshtastic --set range_test.save true  
meshtastic --set range_test.save false
```

Enable range test sender (send every 60 seconds)

```
meshtastic --set range_test.sender 60
```

Disable range test sender

```
meshtastic --set range_test.sender 0
```

Web



All range test module config options are available in the Web UI.

Application Examples

Google Earth Integration

Steps:

Download and open Google Earth

- i. Select File > Import
- ii. Select CSV
- iii. Select Delimited, Comma
- iv. Make sure the button that states “This dataset does not contain latitude/longitude information, but street addresses” is unchecked
- v. Select “rx lat” & “rx long” for the appropriate lat/lng fields
- vi. Click finish

When it prompts you to create a style template, click yes.

- i. Set the name field to whichever column you want to be displayed on the map (don’t worry about this too much, when you click on an icon, all the relevant data appears)
- ii. Select a color, icon, etc. and hit OK.

Your data will load onto the map, make sure to click the checkbox next to your dataset in the sidebar to view it.

My Maps

You can use My Maps. It takes CSVs and the whole interface is much easier to work with.

Google has instructions on how to do that here.

You can style the ranges differently based on the values, so you can have the pins be darker if the SNR or RSSI (if that gets added) is higher.

Openstreetmap - uMap

For an open source solution you could use uMap. A service based on the popular openstreetmap project.

Visit uMap in your preferred language.

Click on "Create a map"

Click on the "Import data"-Icon or press CRTL + I.

Choose your rangetest.csv and click "Import". The dataformat is recognized and the locations are imported onto the map.

Remote Hardware Module Usage

The Remote Hardware Module allows to read, write and watch GPIO pins on a remote node. Config options are: Enabled.

 **INFO**

While configuring this module may be available in clients, setting and reading GPIO's is currently only possible using the Meshtastic Python CLI

Remote Hardware Config Values

Enabled

Whether the module is enabled.

Remote Hardware Module Client Availability

Android

 **INFO**

All Remote Hardware Module config options are available for

Android in app.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Remote Hardware**

Apple

ⓘ INFO

All Remote Hardware Module config options are available on iOS, iPadOS and macOS app and higher at Settings > Module Configuration > Remote Hardware

CLI

ⓘ INFO

All Remote Hardware Module config options are available in the python CLI.

Web

Not implemented.

Remote Hardware Module Usage

⚠ WARNING

GPIO access is fundamentally dangerous because invalid options can physically damage or destroy your hardware. Ensure that you fully understand the schematic for your particular device before trying this as we do not offer a

warranty. Use at your own risk.

Supported Operations

Set any GPIO

Read any GPIO

Receive a notification over the mesh if any GPIO changes state.

Note that it cannot detect fast changes like button presses. For this, look at the Detection Sensor module instead.

The result of reading a GPIO or notifications of GPIO changes will also be sent over MQTT (if enabled) in JSON format (if enabled).

Setup

You can get the latest python tool/library with `pip3 install --upgrade meshtastic` on Windows/Linux/OS-X. See the python section for more details.

To prevent access from untrusted users, you must first make a `gpio` channel that is used for authenticated access to this feature. You'll need to install this channel on both the local and remote node.

The procedure using the python command line tool is:

Connect local device via USB

Create a GPIO channel:

```
meshtastic --ch-add gpio
```

Check the channel has been created and copy the long "Complete URL" that contains all the channels on that device:

```
meshtastic --info
```

Connect the remote device via USB (or use the remote admin feature to reach it through the mesh)

Set it to join the gpio channel you created:

```
meshtastic --seturl theurlyoucopiedinstep3
```

Now both devices should be able to talk over the `gpio` channel.

Send a text message from one to the other to verify. Also run `--nodes` to verify the second node shows up.

Masks

A mask is used to set the GPIOs to control. For GPIO 1, bit 1 of the mask is set (0x2 in hexadecimal), for GPIO 2, bit 2 of the mask is set (0x4 in hexadecimal), and so on. To determine the appropriate mask for the pin(s) that you want to know, the python program (and output) below might help:

```
>>> for i in range(1,45):
...     print(f'GPIO:{i} mask:{hex(2**i)}')
```

Using GPIOs from the Python CLI

ⓘ INFO

You can also control or watch GPIOs of your USB-connected node by setting `--dest` to the local node's ID. No `gpio` channel is needed in this case.

Writing a GPIO

Example: turning 'on' GPIO4

```
meshtastic --port /dev/ttyUSB0 --gpio-wrb 4 1 --
dest 28979058
# Connected to radio
# Writing GPIO mask 0x10 with value 0x10 to !28979058
```

Reading a GPIO

Example: read GPIO4

```
meshtastic --port /dev/ttyUSB0 --gpio-rd 0x10 --dest
28979058
# Connected to radio
# Reading GPIO mask 0x10 from !28979058
# GPIO read response gpio_value=16
```

 **NOTE**

If the mask and the gpio_value match, then the value is "on".

If the gpio_value is 0, then the value is "off".

Watching for GPIO Changes

Example: watching GPIO4 for changes

```
meshtastic --port /dev/ttyUSB0 --gpio-watch 0x10 --  
dest 28979058  
# Connected to radio  
# Watching GPIO mask 0x10 from !28979058  
# Received RemoteHardware typ=GPIO_CHANGED,  
gpio_value=16  
# Received RemoteHardware typ=GPIO_CHANGED,  
gpio_value=0  
# Received RemoteHardware typ=GPIO_CHANGED,  
gpio_value=16  
# < press ctrl-c to exit >
```

Testing GPIO Operations

You can programmatically do operations from your own python code by using the Meshtastic `RemoteHardwareClient` class. See the Python API documentation for more details.

You can add a simple LED and resistor to validate that the GPIO

operations work as expected. Use this tutorial as a guide.

Requirements

- (x2) Meshtastic devices (one device could be on a local computer, and the other one just has to be powered and is the one with the LED to be connected to it)
- (x2) wires (black and yellow; they can be any color but typically black is used for ground)
- (x1) LED
- (x1) 220Ω resistor (somewhat optional, but recommended)
- (x1) Breadboard (optional)

Preparation

Disconnect the remote device from power (battery/usb)

Connect the resistor to the longer (positive) lead of the LED and the yellow wire to the other end of the resistor

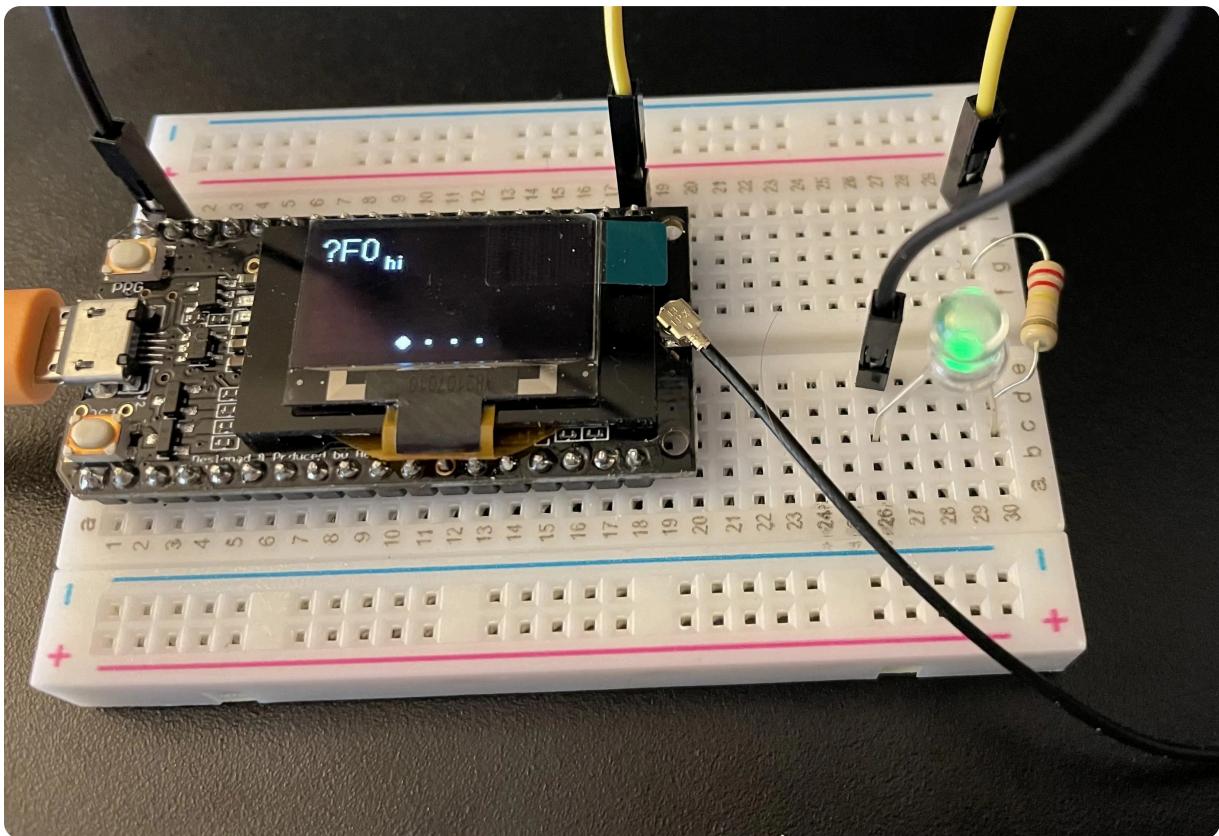
Connect the other end of the yellow wire to a GPIO pin that will not cause any issues (ex: for TLoraV1, we can use GPIO21)

Connect the black "ground" wire from the ground pin on the device (ex: for TLoraV1 it is the end pin next to the RST button) to the shorter (negative) lead of the LED

Power on the device

Validation

By default, the pin may be "off" or "on". (It will most likely "off".) See the steps below for running commands. In the example of GPIO21, the mask would be `0x200000`.



Serial Module Configuration

The serial module config options are: Enabled, Echo, Mode, Receive GPIO, Transmit GPIO, Baud Rate, Timeout, and Override Console Serial Port. Serial Module config uses an admin message sending a `ConfigModule.Serial` protobuf.

This is an interface to talk to and control your Meshtastic device over a serial port. The module can be set to different modes, each fulfilling a different use-case.

```
[16:01:47.297] Connected  
JET: Beep beep I'm a jeep  
[Hello  
JET: Toot toot I'm a boat
```

Serial Module Config Values

Enabled

Enables the serial module.

Echo

If set, any packets you send will be echoed back to your device.

Mode

Defaults to 'Simple'.

Available Values:

`DEFAULT` equals `SIMPLE`.

`SIMPLE` operates as a dumb UART tunnel. What goes in will come out, requires a channel named 'serial'.

`PROTO` exposes the Protobuf Client API on this serial port. You can use this to connect from another device, see the Arduino client library and the API Reference.

`TEXTMSG` will allow you to send a string over the serial port to the device, which will be broadcasted as a text message to the default

channel. Any text message received from the mesh will be sent to the serial port as follows: <Short Name>: <text>.

NMEA will output a NMEA 0183 Data stream containing the internal GPS or fixed position and other node locations as Waypoints (WPL).
CALTOPO will output NMEA 0183 Waypoints (WPL) every 10 seconds for all valid node locations, to be consumed by CalTopo / SARTopo.

Receive GPIO Pin

Set the GPIO pin to the RXD pin you have set up.

Transmit GPIO Pin

Set the GPIO pin to the TXD pin you have set up.



Connect the TX pin to the other device's RX pin, and vice versa. Connect their grounds to each other (not necessary if they're both plugged into the same USB power source.)

Baud Rate

The serial baud rate.

Timeout

The amount of time to wait before we consider your packet as "done".

Override Console Serial Port

If set to true, this will allow Serial Module to control (set baud rate) and use the primary USB serial bus for output. This is only useful for NMEA and CalTopo modes and may behave strangely or not work at all in other modes. Setting TX/RX pins in the Serial Module config will cause this setting to be ignored.



Once module settings are changed, a **reset** is required for them to take effect.

Serial Module Config Client Availability

Android



Serial Module Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio**

Configuration > Serial

Apple

ⓘ INFO

All serial module config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > Serial.

CLI

ⓘ INFO

All serial module config options are available in the python CLI.
Example commands are below:

Setting	Acceptable Values	Default
serial.enabled	true, false	false
serial.echo	true, false	false
serial.mode	DEFAULT SIMPLE PROTO TEXTMSG, NMEA, CALTPO	DEFAULT

Setting	Acceptable Values	Default
serial.rxd	GPIO Pin Number 1-39	Default of <code>0</code> is Unset
serial.txd	GPIO Pin Number 1-33	Default of <code>0</code> is Unset
serial.baud	<div style="display: flex; justify-content: space-around;"> BAUD_DEFAULT BAUD_110 BAUD_300 BAUD_600 BAUD_1200 BAUD_2400 BAUD_4800 BAUD_9600 BAUD_19200 BAUD_38400 BAUD_57600 BAUD_115200 BAUD_230400 BAUD_460800 BAUD_576000 </div>	BAUD_DEFAULT (38400)

Setting	Acceptable Values	Default
	BAUD_921600	
serial.timeout	integer (milli seconds)	Default of 0 corresponds to 250 ms
serial.override_console_serial_port	true, false	false

TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set serial.enabled true --set
serial.echo true
```

Enable / Disable Module

```
meshtastic --set serial.enabled true
meshtastic --set serial.enabled false
```

Enable / Disable Echo

```
meshtastic --set serial.echo true  
meshtastic --set serial.echo false
```

Set Mode

```
meshtastic --set serial.mode DEFAULT  
meshtastic --set serial.mode PROTO
```

Set RXD to GPIO pin number 7

```
meshtastic --set serial.rxd 7
```

Set TXD to GPIO pin number 28

```
meshtastic --set serial.txd 28
```

Set Baud Rate

```
meshtastic --set serial.baud BAUD_DEFAULT  
meshtastic --set serial.baud BAUD_576000
```

Set Timeout to 15 milli seconds

```
meshtastic --set serial.timeout 15
```

Web



INFO

All serial module config options are available in the Web UI.



WARNING

GPIO access is fundamentally dangerous because invalid options can physically damage or destroy your hardware. Ensure that you fully understand the schematic for your particular device before trying this as we do not offer a warranty. Use at your own risk.

This module requires attaching a peripheral accessory to your device. It will not work without one.

Examples

Default is to use RX GPIO 16 and TX GPIO 17.

Basic Usage

Enable the module by setting `serial.enabled` to `1`.

Set the pins (`serial.rxd` / `serial.txd`) for your preferred RX and TX GPIO pins. On tbeam boards it is recommended to use:
RXD 13

TXD 14

Set `serial.timeout` to the amount of time to wait before we consider your packet as "done".

(Optional) set `serial.mode` to `TEXTMSG` if you want to send messages to/from the general text message channel. For more specific control, use the `PROTO` mode, e.g. in combination with the Arduino client library.

Connect to your device over the serial interface at `38400 8N1`.

With tio - `tio -e -b 38400 -f none /dev/myserialport`

Send a packet up to 237 bytes in length. This will get broadcasted over the default channel.

(Optional) Set `serial.echo` to `1` and any message you send out will be echoed back to your device.

Interfacing PIR Sensor With External Microcontroller

The following are examples of using either a Raspberry Pi Pico or Arduino Mini Pro connected to a PIR sensor to detect motion. When motion is detected, a message is sent via serial to the T-Beam. The T-Beam transmits the message as text over the default channel by utilizing the serial module in `TEXTMSG` mode.

Meshtastic Software Configuration

Serial module enabled, mode: TEXTMSG

GPIO Pins (For T-Beam) RX 13, TX 14

38400 Baud

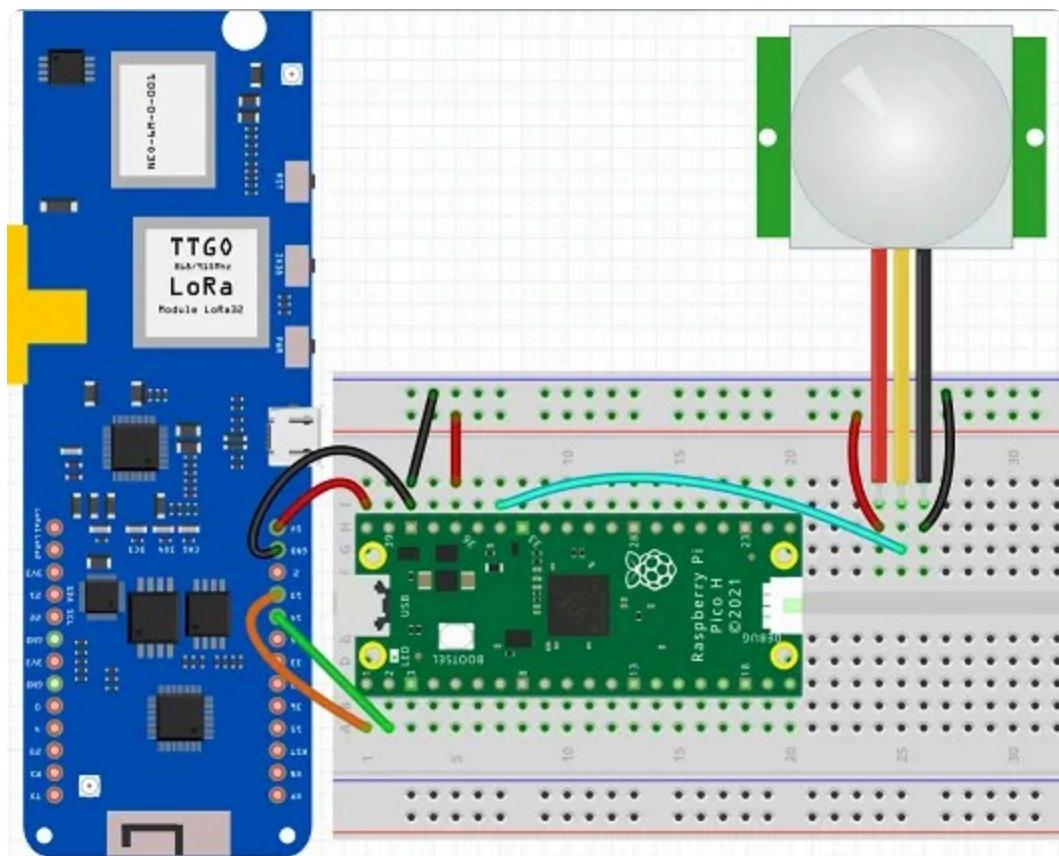
Raspberry Pi Pico BOM

A Raspberry Pi Pico running CircuitPython

T-Beam V1.1 running Meshtastic

PIR Sensor (Adafruit Breadboard Model)

Raspberry Pi Pico Wiring



TX pin 14 on the T-Beam to RX pin 2 on the Pico
RX pin 13 on the T-Beam to TX pin 1 on the Pico
PIR sensor Vcc and GND pins to Vcc/GND on breadboard
respectively
PIR sensor trigger line to Pico GPIO28 (Pico pin 34)
GND pin on T-Beam to GND pin 38 on the Pico
GND pin 38 on the Pico to breadboard ground rail
3V3 pin 36 on the Pico to the breadboard positive voltage rail
Optional, to power the Pico off of the T-Beam instead of USB:
Connect 5V pin on T-Beam to Vbus pin 40 on the Pico

Circuit Python Code

```
import time
import board
import busio
import digitalio

# Setup PIR sensor on GP28
pir = digitalio.DigitalInOut(board.GP28)
pir.direction = digitalio.Direction.INPUT

# Setup serial UART connection TX/RX on (GP0/GP1)
uart = busio.UART(board.GP0, board.GP1,
baudrate=38400, timeout=0)

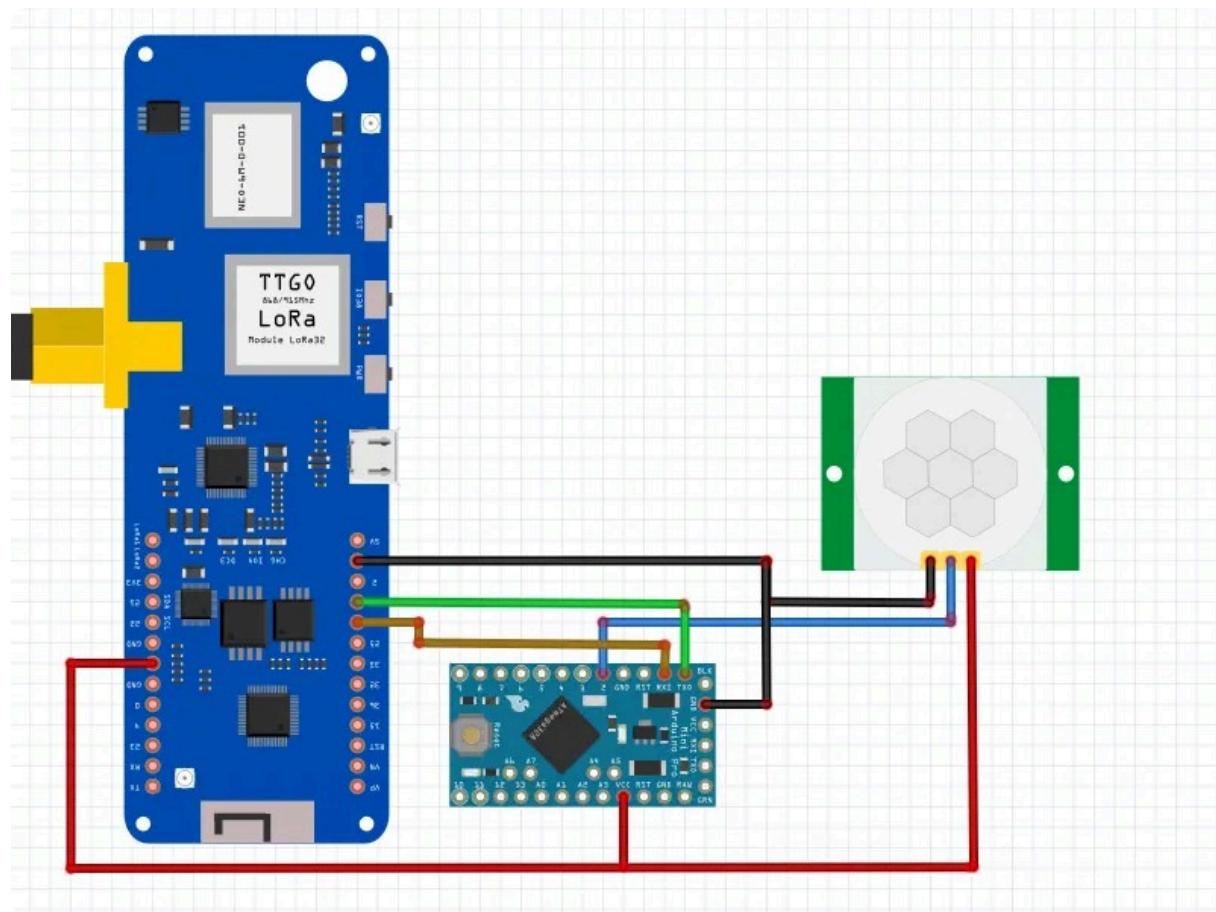
while True:
    if(pir.value == True):
        uart.write(bytes("Motion Detected", "ascii"))
```

Arduino Mini Pro BOM

An Arduino Mini Pro with example sketch from below uploaded to it.

T-Beam V1.1 running Meshtastic
PIR Sensor (Adafruit Breadboard Model)

Arduino Mini Pro Wiring



T-BEAM RX PIN 13 to TX PIN on the ARDUINO MINI

T-BEAM TX PIN 14 to RX PIN on the ARDUINO MINI

T-BEAM PIN 3.3V to 3.3V PIN on the ARDUINO MINI

T-BEAM PIN GND to GND PIN on the ARDUINO MINI
ARDUINO MINI PIN 2 to OUT PIN on the PIR SENSOR
ARDUINO MINI PIN 3.3V to 3.3V on the PIR SENSOR
ARDUINO MINI PIN GND to GND PIN on the PIR SENSOR

Arduino Mini Pro Code

```
int LED = 13; // the pin to which the LED is
connected
int PIR = 2; // the pin to which the sensor is
connected
int previousState = LOW; // previous state of the
sensor

void setup() {
pinMode(LED, OUTPUT); // initialize the LED as an
output
pinMode(PIR, INPUT); // initialize the sensor as an
input
Serial.begin(9600); // initialize serial
communication
}

void loop(){
    int currentState = digitalRead(PIR); // read the
current state of the sensor
    if (currentState != previousState) { // check if
the state has changed
        if (currentState == HIGH) { // check if there is
motion
            digitalWrite(LED, HIGH); // turn the LED on
            Serial.println("Motion Detected");
        }
    }
}
```


Store & Forward Module Settings

Overview

Using this module, a client device can ask a special Store & Forward Router to resend text messages after the client has been temporarily not in LoRa range of the mesh.

ⓘ INFO

Only ESP32 based devices with onboard PSRAM like the T-Beam and T3S3 can be a Store & Forward Router. Requires the device to use at least firmware version 2.2.23 and to be set as a `ROUTER` or `ROUTER_CLIENT`.

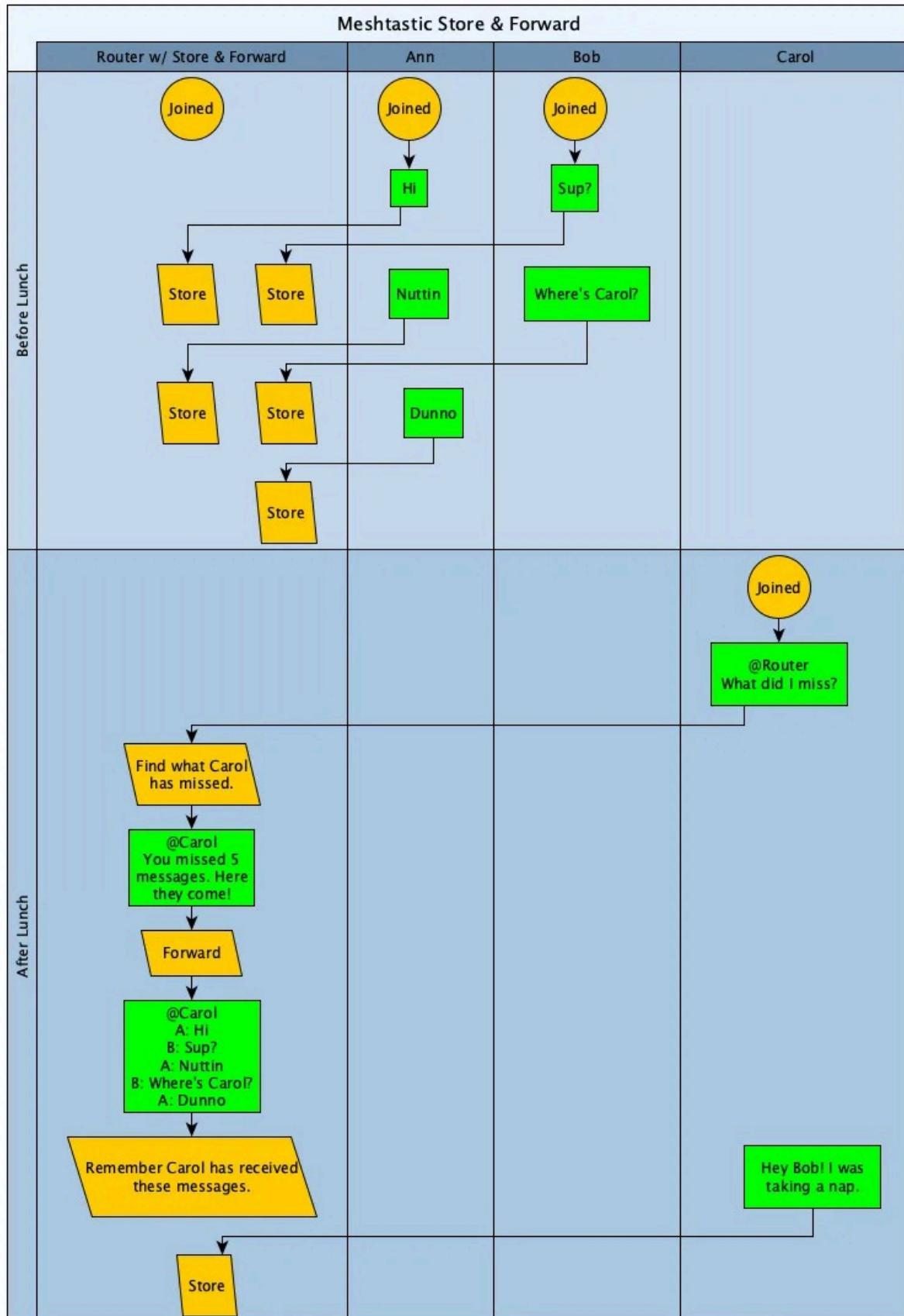
When a client device requests the history from the Store & Forward Router, the router will resend the text messages over LoRa that it has received. The router will only return messages that are within the time window the client has requested up to the maximum number of messages configured for the router. The router does not know which messages the client device actually missed, so it is possible that you receive duplicates.

ⓘ IMPORTANT

Be mindful when requesting the history, as the router might send a lot of messages which will burden your mesh for a short period of time.

Details

How it works



Requirements

Initial requirements for the Store and Forward Router:

Must be installed on a `ROUTER` or `ROUTER_CLIENT` node.

This is an artificial limitation, but is in place to enforce best practices.

Router nodes are intended to be always online. If this module misses any messages, the reliability of the stored messages will be reduced.

ESP32 Processor based device with onboard PSRAM (T-Beam > v1.0, T3S3, and maybe others).

Usage Overview

To use / test this you will want at least 3 devices

One ESP32 device with PSRAM configured as `ROUTER` or `ROUTER_CLIENT`.

Two others will be regular clients. If one client sends a text message when the other is not in range, the other can request the history from the router to receive the missed message when it is back in range.

Router setup

Configure your device as a `ROUTER` or `ROUTER_CLIENT`.

Name your router node something that makes it easily identifiable, aka "Router".

Configure the Store and Forward module

Required - Enable the module

```
meshtastic --set store_forward.enabled true
```

Optional - Disable sending heartbeat.

```
meshtastic --set store_forward.heartbeat false
```



TIP
Best to disable the heartbeat (which is sent every 15 minutes) when all client devices have identified the router to reduce network traffic.

Client Usage

Currently implemented in the Android and Apple apps version 2.2.23 and higher. To request the history from the Store & Forward Router, for Android it is required to send it a direct message containing the text "SF" (without quotes). The router will then respond with the requested messages. The Apple apps will also show whether a node is a Store & Forward Router in the node list

after it heard the heartbeat. You can then long press the node and select "Client History" to request the history from the router.

Settings

Enabled

Enables the module.

Heartbeat

The Store & Forward Router sends a periodic message onto the network. This allows connected devices to know that a router is in range and listening to received messages. A client like Android, iOS, or Web can (if supported) indicate to the user whether a store and forward router is available.

History Return Max

Sets the maximum number of messages to return to a client device when it requests the history.

History Return Window

Limits the time period (in minutes) a client device can request.

Records

Set this to the maximum number of records the router will save.

Best to leave this at the default (0) where the module will use 2/3 of your device's available PSRAM. This is about 11,000 records.

Store & Forward Module Config Client Availability

Android

INFO

Store and Forward Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Store & Forward**

Apple

INFO

All Store & Forward module config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > Store & Forward.

CLI

Setting	Acceptable Values	Default
store_forward.enabled	true, false	false
store_forward.heartbeat	true, false	false
store_forward.history_return_max	integer	0 (25 messages)
store_forward.history_return_window	integer	0 (240 minutes)
store_forward.records	integer	0 (\approx 11,000 records)

 **TIP**

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set store_forward.enabled true --  
set store_forward.history_return_max 0
```

Examples of CLI Usage

Enable the module

```
meshtastic --set store_forward.enabled true
```

Disable the module

```
meshtastic --set store_forward.enabled false
```

Set store_forward.heartbeat to default

```
meshtastic --set store_forward.heartbeat 0
```

Set store_forward.history_return_max to default (25 messages)

```
meshtastic --set store_forward.history_return_max 0
```

Set store_forward.history_return_max to 100 messages

```
meshtastic --set store_forward.history_return_max 100
```

Set `store_forward.history_return_window` to default (240 minutes)

```
meshtastic --set store_forward.history_return_window  
0
```

Set `store_forward.history_return_window` to 1 day (1440 minutes)

```
meshtastic --set store_forward.history_return_window  
1440
```

Set `store_forward.records` to default (\approx 11,000 records)

```
meshtastic --set store_forward.records 0
```

Set `store_forward.records` to 100 records

```
meshtastic --set store_forward.records 100
```

Web

 **INFO**

Store and Forward configuration is not currently available via the web client.

Telemetry Module Configuration

The Telemetry Module provides three types of data over the mesh: Device metrics (Battery Level, Voltage, Channel Utilization and Airtime) from your Meshtastic device, Environment Metrics from attached I2C sensors, and Air Quality Metrics from attached I2C particle sensors.

Supported sensors connected to the I2C bus of the device will be automatically detected at startup. Environment Telemetry and Air Quality must be enabled for them to be instrumented and their readings sent over the mesh.

Currently Supported Sensor Types

Sensor	I ² C Address	Data Points
BMP280	0x76, 0x77	Temperature and barometric pressure
BME280	0x76, 0x77	Temperature, barometric pressure and humidity
BME680	0x76, 0x77	Temperature, barometric pressure, humidity and air resistance
MCP9808	0x18	Temperature
INA260	0x40, 0x41	Current and Voltage
INA219	0x40, 0x41	Current and Voltage
LPS22	0x5D, 0x5c	Barometric pressure
SHTC3	0x70	Temperature and humidity

Sensor	I²C Address	Data Points
SHT31	0x44	Temperature and humidity
PMSA003I	0x12	Concentration units by size and particle counts by size

Module Config Values

Environment Telemetry Enabled

Enable the Environment Telemetry (Sensors).

Environment Metrics Update Interval

How often we should send Environment(Sensor) Metrics over the mesh.

Default is `900` seconds (15 minutes).

Device Metrics Update Interval

How often we should send Device Metrics over the mesh.

Default is `900` seconds (15 minutes).

Device Metrics to a connected client app will always be sent once per minute, regardless of this setting.

Environment Screen Enabled

Show the environment telemetry data on the device display.

Default is `false`.

Display Fahrenheit

The sensor is always read in Celsius, but the user can opt to display in Fahrenheit (on the device display only) using this setting.

Default is `false`.

Air Quality Enabled

This option is used to enable/disable the sending of air quality metrics from an attached supported sensor over the mesh network.

Default is `false`.

Air Quality Interval

This option is used to configure the interval (in seconds) that should be used to send air quality metrics from an attached

supported sensor over the mesh network.

Default is `900` seconds (15 minutes).

Telemetry Config Client Availability

Android

 **INFO**

Telemetry Config options are available for Android.

Open the Meshtastic App

Navigate to: **Vertical Ellipsis (3 dots top right) > Radio Configuration > Telemetry**

Apple

 **INFO**

All telemetry module config options are available on iOS, iPadOS and macOS at Settings > Module Configuration > Telemetry (Sensors).

CLI

 **INFO**

All telemetry module config options are available in the python CLI. Example commands are below:

Settings

Setting	Acceptable Values	Default
telemetry.device_update_interval	integer (seconds)	Default 0 is 15 minutes(900 seconds).
telemetry.environment_display_fahrenheit	true, false	false
telemetry.environment_measurement_enabled	true, false	false
telemetry.environment_screen_enabled	true, false	false
telemetry.environment_update_interval	integer (seconds)	Default 0 is 15 minutes(900 seconds).
telemetry.air_quality_enabled	true, false	false

Setting	Acceptable Values	Default
telemetry.air_quality_interval	integer (seconds)	Default 0 is 15 minutes(900 seconds).

TIP

Because the device will reboot after each command is sent via CLI, it is recommended when setting multiple values in a config section that commands be chained together as one.

Example:

```
meshtastic --set telemetry.device_update_interval 0 --set telemetry.environment_update_interval 0
```

Set module update intervals (Default of 0 is 5 Minutes)

```
meshtastic --set telemetry.device_update_interval 0
// Device Metrics Two Minutes
meshtastic --set telemetry.device_update_interval 0
// Environment Metrics Two Minutes
meshtastic --set
```

Enable/Disable Environment Module

```
meshtastic --set  
telemetry.environment_measurement_enabled true  
meshtastic --set  
telemetry.environment_measurement_enabled false
```

Enable/Disable on device screen

```
meshtastic --set telemetry.environment_screen_enabled  
true  
meshtastic --set telemetry.environment_screen_enabled  
false
```

Enable / Disable Display Fahrenheit

```
meshtastic --set  
telemetry.environment_display_fahrenheit true  
meshtastic --set  
telemetry.environment_display_fahrenheit false
```

Web

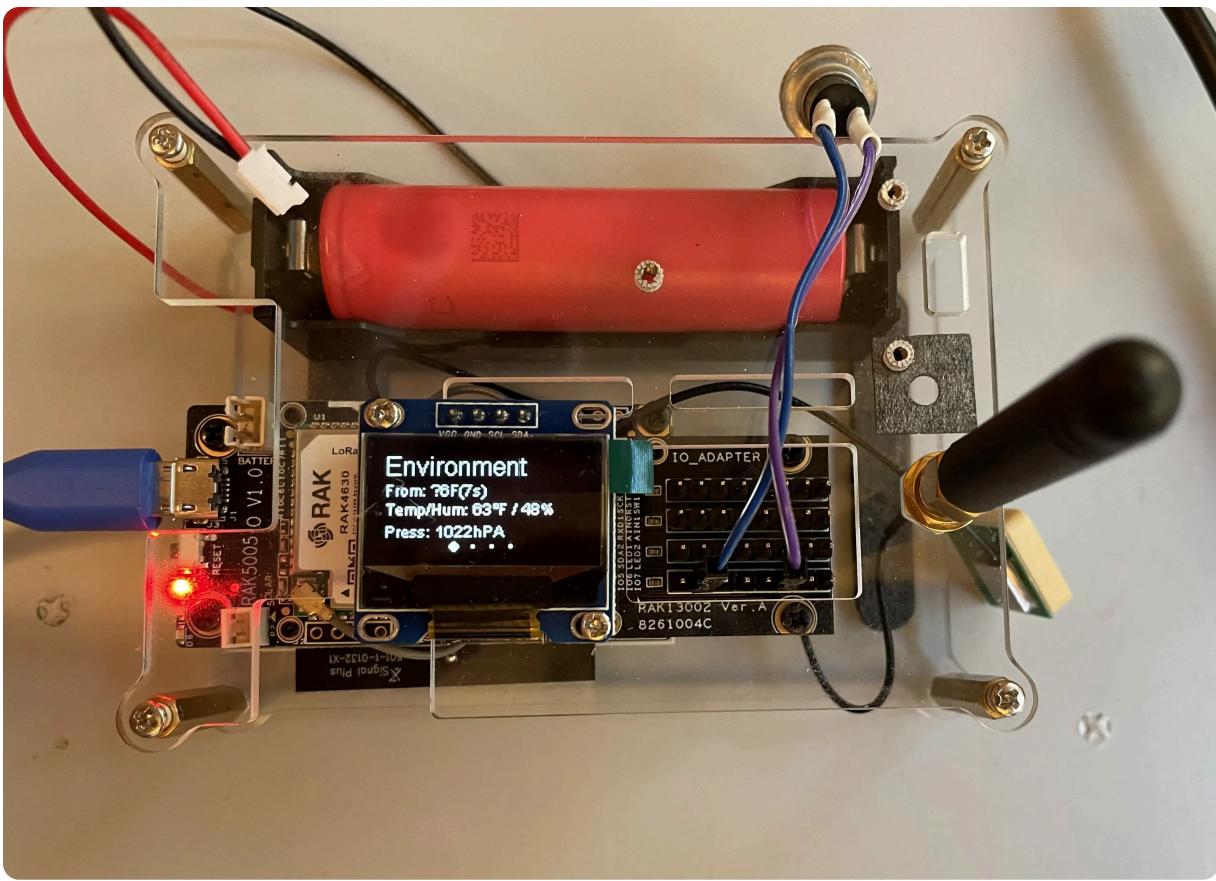
INFO

All telemetry module config options are available in the Web UI.

Examples

RAK 4631 with BME680 Environment Sensor

Setup of a RAK 4631 with Environment Sensor



Requirements:

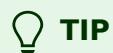
RAK4631

Environment Sensor

Steps:

configure the device:

```
meshtastic --set  
telemetry.environment_measurement_enabled true --set  
telemetry.environment_screen_enabled true --set  
telemetry.environment_display_fahrenheit true
```



TIP
While the above values serve as an example and can be modified to fit your specific needs, it is advisable to chain multiple commands together, as demonstrated in the example. This approach will minimize the number of necessary reboots.

Device will reboot after command is sent.

When the device boots again it should say "Telemetry" and it may show the sensor data

If this does not appear to have any effects, run:

```
meshtastic --noprotocol
```

And examine the serial logs for Telemetry diagnostic information.

Supporting Additional Sensors

Environment Metrics

The environment metrics in the telemetry module supports a limited amount of fields as they are stored in memory on the device. Support for sensors that provide one or more of the following fields can potentially be added to the main firmware provided there is a GPL licensed library for us to include to support it, and the library size is not prohibitive.

Temperature

Relative Humidity

Barometric Pressure

Gas Resistance (AQI)

Voltage

Current

Supporting Other Sensor types

For other interesting sensor types and use cases we need to add a portnum for more generic telemetry packets and a second MCU will be required to interact with the sensor and process the data to be sent over the mesh. This data will not be stored in the nodedb on the device.

Traceroute Module Usage

Overview

Due to the limited bandwidth of LoRa, Meshtastic does not keep track of the nodes a message used to hop to the destination. However, from firmware 2.0.8 on, there is a Traceroute Module that can show you this.

Only nodes that know the encryption key of the channel you use can be tracked. Also note that a message may arrive via multiple routes due to duplication because of rebroadcasting. This module will only track the hops of the first packet containing the traceroute request that arrived at the destination.

In order to use it, make sure your devices use firmware version 2.0.8 or higher.

Repeater Behavior

Repeater nodes will appear in the traceroute log if they have the encryption key, but if they are not in the node list they might appear as "Unknown" depending on the app.

Traceroute Module Client Availability

Android

Within the node list, long hold a destination node and select 'Traceroute' to send the request. Depending on the amount of hops that is needed, this might take a while. The result will be shown using a pop-up.

Apple

Within the node list, long hold a destination node and select 'Trace Route' to send the request. Depending on the amount of hops, this might take a while. On iOS/iPadOS 17 and macOS 14+, the results will be shown in the node's Trace Route Log in the node details. For earlier versions, check Settings > Logging > Mesh Log.

CLI

Make sure the CLI is at least version 2.0.6. Then use this command:

```
CLI traceroute command
```

```
meshtastic --traceroute 'destinationId'
```

Where for `destinationId` you have to fill in the ID of the destination you want to track the hops to, which you can get from running `meshtastic --nodes`. Depending on your OS, you might need quotation marks around the ID. Then it will send a specific message to track the hops. For example, this is what you will get:

```
Traceroute from !25048234 to !bff18ce4
```

```
meshtastic --traceroute '!bff18ce4'  
Connected to radio  
Sending traceroute request to !bff18ce4 (this could  
take a while)  
Route traced:  
!25048234 --> !ba4bf9d0 --> !bff18ce4
```

The first ID shown is the device you are connected to with the CLI. As you can see, this packet went through one other node to get to its destination.

Web

Not yet implemented.

Remote Node Administration

DISCLAIMER

This is an advanced feature that few users should need. Keep in mind that it is possible (if you are not careful) to assign settings to a remote node that cause it to completely drop off of your mesh. We advise network admins have a test node to test settings with before applying changes to a remote node to prevent this.

This feature allows you to remotely administer Meshtastic nodes through the mesh.

Prerequisites

For any node that you want to administer, you must:

Configure a channel with the channel name of `admin`.

Set the PSK for the `admin` channel with the same key you wish to administer other all other nodes of the mesh with.

Creating the admin channel

By default, nodes will **only** respond to administrative commands via the local USB/Bluetooth/TCP interface. This provides basic security to prevent unauthorized access and is how normal administration and settings changes work. The only difference for the remote case is that we are sending those commands over the mesh.

Before a node will allow remote admin access, it must have a primary channel:

Command

```
meshtastic --info
```

Expected output

```
Connected to radio
```

```
...
```

```
Channels:
```

```
PRIMARY psk=default { "psk": "AQ==" }  
Primary channel URL: https://meshtastic.org/  
e/#CgMSAQESCggBOANAA0gBUBS
```

From this output you see can that this node knows about only one

channel and that its PSK is set to the default value.

Now we add an admin channel:

 **NOTE**

The name of the channel is important and must be `admin`.

Command

```
meshtastic --ch-add admin
```

Expected output

```
Connected to radio  
Writing modified channels to device
```

Run `meshtastic --info` and your channels will now look like this:

Expected output

```
Connected to radio
```

```
...
```

```
Channels:
```

```
PRIMARY psk=default { "psk": "AQ==" }  
SECONDARY psk=secret { "psk": "YyDCitupTA00XTcaMDxyNhDpPa3eThiC  
"name": "admin" }  
Primary channel URL: https://meshtastic.org/e/#CgMSAQESCggBOANAA0
```

Notice that now we have a new secondary channel and the `--info` option prints out TWO URLs. The `Complete URL` includes all of the channels this node understands. The URL contains the preshared keys and should be treated with caution and kept a secret. When deploying remote administration, you only need the node you want to administer and the node you are locally connected to, to know this new "admin" channel. All of the other nodes will forward the packets as long as they are a member of the primary channel.

Sharing the admin channel with other nodes

Creating an "admin" channel automatically generates a preshared key. In order to administer other nodes remotely, we need to copy the preshared key to the new nodes.

For this step you need physical access to both the nodes.

Create the "admin" channel on the "local node" using the instructions above.

Copy the "Complete URL" someplace for permanent reference/access.

Connect to the "remote node" over the USB port.

For the "remote node" type

```
meshtastic --seturl the-url-from-step-2
```

Run `meshtastic --info` and confirm that the "Complete URL" is the same for both of the nodes.

At this point you can take your remote node and install it far away. You will still be able to change any of its settings.

Remotely administering your node

Now that both your local node and the remote node contain your secret admin channel key, you can now change settings remotely:

Get the node list from the local node:

Command

```
meshtastic --nodes
```

Expected output

Connected to radio

/-----\-----

\

N	User	AKA	ID	Latitude	Longitude	Altitude	Battery

Using the node ID from that list, send a message through the mesh telling that node to change its owner name.

 **INFO**

The --dest argument value must be in single quotes for linux/mac: '!28979058', no quotes for Windows: !28979058.

Command

```
meshtastic --dest '!28979058' --set-owner "Im Remote"
```

Expected output

```
Connected to radio
Setting device owner to Im Remote
Waiting for an acknowledgment from remote node (this
could take a while)
```

And you can now confirm via the local node that the remote node has changed:

Command

```
meshtastic --nodes
```

Expected output

```
Connected to radio
/-----
 \
|N| User | AKA| ID | Position | Battery| SNR
| |
|-----+----+-----+-----+-----+
| |
|1|Im Remote|IR |!28979058|25.0382°, 121.5731°, N/A| N/A | 8.75
ago|
\-----
 /
```

Note: you can change **any** parameter, add channels or get settings from the remote node. Here's an example of setting ls_secs using the `--set` command and printing the position settings from the remote node using the `--get` command:

Command

```
meshtastic --dest '!28979058' --set power.ls_secs
301 --get position
```

Expected output

```
Connected to radio
Requesting current config from remote node (this can
take a while).
```

To set the `hop_limit` to 4 and then get the lora settings to confirm your new settings:

Command

```
meshtastic --dest '!28979058' --set lora.hop_limit 4  
--get lora
```

Expected output

```
lora:  
use_preset: true  
region: US  
hop_limit: 3  
tx_enabled: true  
tx_power: 30
```

```
Set lora.hop_limit to 4  
Writing modified preferences to device  
Requesting current config from remote node (this can  
take a while).  
Received an ACK.  
Completed getting preferences  
Waiting for an acknowledgment from remote node (this  
could take a while)
```

```
lora:  
use_preset: true  
region: US
```

Admin Channel Setup is Complete

You've finished setting up and adding two devices to the admin channel. Remember, because this is a mesh network, it doesn't matter which node you are at; you could administer your first device we set up from the second one we added to the channel. And the settings and examples on this page are just a taste of the other settings you can set.

For further reading, I recommend starting out with the Meshtastic Python CLI Guide if you haven't already gone through this (hopefully you have since you are reading this). But for a full reference to the settings you can change, please see:

[Settings Overview and Complete list of user settings in Protobufs](#)

Configuration Tips

Roles

Leave your ROLE set to `CLIENT` unless you're sure another role would suit the node's purpose. All `CLIENT` nodes will "repeat" and "route" packets. Roles are for very specific applications so please read the documentation before changing your node's role.



INFO

Users report that the current implementation of the `ROUTER` role with bluetooth switched off causes instability. If you would like the functionality associated with this role, it is recommended you choose `ROUTER_CLIENT` until a firmware fix is issued.

(Not) Sharing Your Location

Telemetry is shared over your PRIMARY channel. This means that if your node has acquired GPS coordinates from an integrated GPS chip, or from your mobile device, your coordinates will be sent to the mesh over this channel, using it's defined encryption (if any).

By default the PRIMARY channel's name is LongFast with the encryption key "AQ==" (Base64 equivalent of Hex 0x01). If this is

left unchanged, your location will be shared with all nodes in range that are also using the default channel.

Creating a Private Primary with Default Secondary

If you'd like to connect with other Meshtastic users but only share your location with trusted parties, you may create a private PRIMARY channel and use the defaults for a SECONDARY channel.

Ensure you have not changed the LoRa Modem Preset from the default `unset` / `LONG_FAST`.

On your PRIMARY channel, set anything you'd like for the channel's name and choose a random PSK.

Enable a SECONDARY channel named "LongFast" with PSK "AQ==".

If your LoRa channel is set to the default (0), the radio's frequency will be automatically changed based on your PRIMARY channel's name. In this case, you will have to manually set it back to your region's default (in LoRa settings) in order to interface with users on the default channel:

Default Primary Channels by Region

US	EU_433	EU_868	CN	JP	ANZ	KR	TW	RU	IN	NZ_865	TI
20	4	1	36	20	20	12	16	2	4	4	16

To quickly test this configuration, find and scan your region's QR from this repository. Remember to generate a new PSK for your private channel before sharing with your trusted nodes.

Rebroadcast "Public" Traffic

Meshtastic nodes will rebroadcast all packets if they share LoRa modem settings, regardless of encryption (unless Rebroadcast mode is set to `LOCAL_ONLY`).

 **INFO**

If you would like your nodes to include/expand the "public" mesh, you must use the default modem preset `LONG_FAST`. If you change your PRIMARY channel name, you must manually set the LoRa channel to the default for your region (see above).

Chat Channels VS LoRa Modem Channels

Meshtastic uses the word "channels" to define two different configuration properties: Messaging Channels & LoRa Modem Channels

Radio Config: Channels

These configure "message groups" and include your PRIMARY and SECONDARY channels. All SECONDARY channels use the same LoRa modem config as your PRIMARY channel (including LoRa channel number).

There are 8 total chat channels. Channel 0 is your PRIMARY channel, with channels 1-7 available for private group messaging and/or special channels such as `admin`.

Radio Config: LoRa: Channel Number

This configures the frequency the radio is set to. Check out the frequency calculator to view the relationship between "channel number" and radio frequency.

Best Practices

If you are part of a large mesh and don't know what a setting does, don't change it (unless you're super curious).

Leave your MAX HOPS set to 3 unless you're sure you need more (or less) to reach your destination node.

TEST your settings and hardware before you install in hard-to-reach locations.

Connecting a node to the public MQTT server may publish the locations of all nodes in your mesh to the internet. This will also add every globally connected node to your node database and potentially flood your mesh with all types of packets.

Supported Hardware

Devices

10 items

Antennas

4 items

Solar Powered

1 items

Devices | Supported Hardware Overview

Supported Devices

Meshtastic firmware can be installed on a wide range of development boards. The list below provides a brief comparison of currently supported hardware.

Which board should I choose?

While all the boards listed on this page will run Meshtastic and mesh with each other, some current community favorites are:

RAK Meshtastic Start Kit

Station G1

LILYGO LoRa T3-S3

HELTEC LoRa V3

Please do your research and choose the board that meets your needs (or maybe already have in a bin somewhere).

 **INFO**

The Semtech SX1262 transceiver is newer than the SX1276 and provides increased receive and transmit performance.

nRF52-based devices use a fraction of the power compared to ESP32-based devices and are therefore generally preferred for solar and handset applications.

ESP32-based devices require more power to operate but are typically lower-cost alternatives that do perform well when using house power, or for handsets that only require a day or two of runtime, and for applications that require WiFi connectivity.

RAK Wisblock

Modular hardware system with Base, Core and Peripheral modules including the low-power and solar ready nRF52840-based Meshtastic Starter Kit (19007 & 4631).

WisBlock Core Modules

Name	MCU	Radio	WiFi	BT	GPS
RAK4631	nRF52840	SX1262	NO	5.0	add-on
RAK11200	ESP32	add-on	2.4GHz b/g/n	4.2	add-on
RAK11310	RP2040	SX1262	NO	NO	add-on

Base Boards

RAK5005-O

RAK19007

RAK19003

RAK19001

WisBlock Displays

RAK1921

RAK1400

WisBlock Peripherals

RAK1910 GPS

RAK12500 GPS

RAK18001 Buzzer

RAK13002 IO

RAK14001 RGB LED

RAK12002 RTC

RAK1901 Temperature and Humidity Sensor

RAK1902 Barometric Pressure Sensor

RAK1906 Environment Sensor

RAK12013 Radar Sensor

RAK13800 Ethernet Module

LILYGO® T-Beam

Boards complete with GPS, 18650 battery holder, and optional screen.

Name	MCU	Radio	WiFi	BT	GPS
T-Beam v0.7	ESP32	SX1276	2.4GHz b/ g/n	4.2	YES
T-Beam v1.1	ESP32	SX1276	2.4GHz b/ g/n	4.2	YES
T-Beam with M8N	ESP32	SX1276 SX1262	2.4GHz b/ g/n	4.2	YES
T-Beam S3-Core	ESP32-S3	SX1262	2.4GHz b/ g/n	5.0	YES
T- BeamSUPREME	ESP32-S3	SX1262	2.4GHz b/ g/n	5.0	YES

LILYGO® T-Echo

All-in-one unit with E-Ink screen, GPS and battery in injection-molded case. Features the low-power nRF52840 for long battery life.

Name	MCU	Radio	WiFi	BT	GPS
T-Echo	nRF52840	SX1262	NO	5.0	YES

LILYGO® LoRa

Inexpensive basic ESP32-based boards.

Name	MCU	Radio	WiFi	BT	GPS
LoRa32 V1	ESP32	SX127x	2.4GHz b/ g/n	4.2	NO
LoRa32 V1.3	ESP32	SX127x	2.4GHz b/ g/n	4.2	NO
LoRa32 V2.0	ESP32	SX127x	2.4GHz b/ g/n	4.2	NO
LoRa32 V2.1-1.6	ESP32	SX127x	2.4GHz b/ g/n	4.2	NO
LoRa32 V2.1-1.8	ESP32	SX1280	2.4GHz b/ g/n	4.2	NO
LoRa32 T3-S3 V1.0	ESP32-S3	SX1262 SX1276 SX1280	2.4GHz b/ g/n	5.0	NO

LILYGO® T-Deck

Standalone device with screen and keyboard

Name	MCU	Radio	WiFi	BT	GPS
T-Deck	ESP32-S3FN8	SX1262	YES	5.0	NO

LILYGO® T-Watch S3

Name	MCU	Radio	WiFi	BT	GPS
T-Watch S3	ESP32-S3	SX1262	YES	5.0	NO

HELTEC® LoRa 32

Inexpensive basic ESP32-based boards.

Name	MCU	Radio	WiFi	BT	GPS
LoRa32 V2.1	ESP32	SX127x	2.4GHz b/g/n	4.2	NO
LoRa32 V3/3.1	ESP32-S3FN8	SX1262	2.4GHz	5.0	NO

Name	MCU	Radio	WiFi	BT	GPS
			b/g/n		
Wireless Stick Lite V3	ESP32-S3FN8	SX1262	2.4GHz b/g/n	5.0	NO
Wireless Tracker	ESP32-S3FN8	SX1262	2.4GHz b/g/n	5.0	YES
Wireless Paper	ESP32-S3FN8	SX1262	2.4GHz b/g/n	5.0	NO

Nano Series

Portable and durable devices designed for Meshtastic.

Name	MCU	Radio	WiFi	BT	GPS
Nano G2 Ultra	NRF52840	SX1262	2.4GHz b/ g/n	5.0	YES
Nano G1 Explorer	ESP32 WROOM	SX1262	2.4GHz b/ g/n	4.2	YES
Nano G1	ESP32	SX1276	2.4GHz b/	4.2	YES

Name	MCU	Radio	WiFi	BT	GPS
	WROOM		g/n		

Station G1

High power LoRa transceiver designed for Meshtastic Licensed HAM operation.

Name	MCU	Radio	WiFi	BT	GPS
Station G1	ESP32 WROOM	SX1262	2.4GHz b/ g/n	4.2	YES

Raspberry Pi Pico

Fast versatile boards using the RP2040.

Name	MCU	Radio	WiFi	BT	GPS
Raspberry Pi Pico	RP2040	SX1262	2.4GHz b/g/n	not supported	NO

Pico Peripherals

SSD1306 OLED Display

SH1106 OLED Display

CardKB Keyboard

RAK WisBlock Devices

The RAK WisBlock is a modular hardware system that can be used to build Meshtastic devices.

RAK Wireless currently sells a Meshtastic Starter kit that has the minimum you need to get started.

If you wish to purchase parts separately, you will need a WisBlock Base Board and a WisBlock Core Module. Please ensure you choose the correct operating frequency for your country when purchasing.

You can optionally purchase peripherals such as a GPS module, Screen, Sensor, or other various modules.

Please see the RAK documentation for the correct way to connect your hardware to the baseboard to ensure that you do not damage the device.

Resources

[RAK's Wisblock Documentation Center](#)

[RAK's GitHub Page for the WisBlock](#)

[Purchase links](#)

See purchase links under specific base boards, core modules, and peripherals

China RAK Direct RAK Wireless Starter Kit w/ Gen2 Base board

US Distributor Rokland RAK Wireless Starter Kit w/ Gen2 Base
board

US Distributor Rokland RAK Wireless Starter Kit w/ Gen1 Base
board

RAK WisBlock Base Boards

Operation requires both a base board and a core module.

RAK5005-O

⚠ CAUTION

The RAK5005-O is no longer in production. It is recommended to use the RAK19007 instead.

RAK5005-O - The original WisBlock Base Board.

Slots

- (x1) Core Module slot
- (x1) WisBlock IO Module slot
- (x4) WisBlock Sensor Module slots

Buttons

- (x1) Reset Button

It may be possible to add a user button using the 13002 IO module.

Connectors

Connector for 3.7v LiPo battery (with charge controller)

Connector for 5v solar panel (max 5.5v)

I²C, UART, GPIOs and analog input accessible with solder contacts

Micro USB port for debugging and power

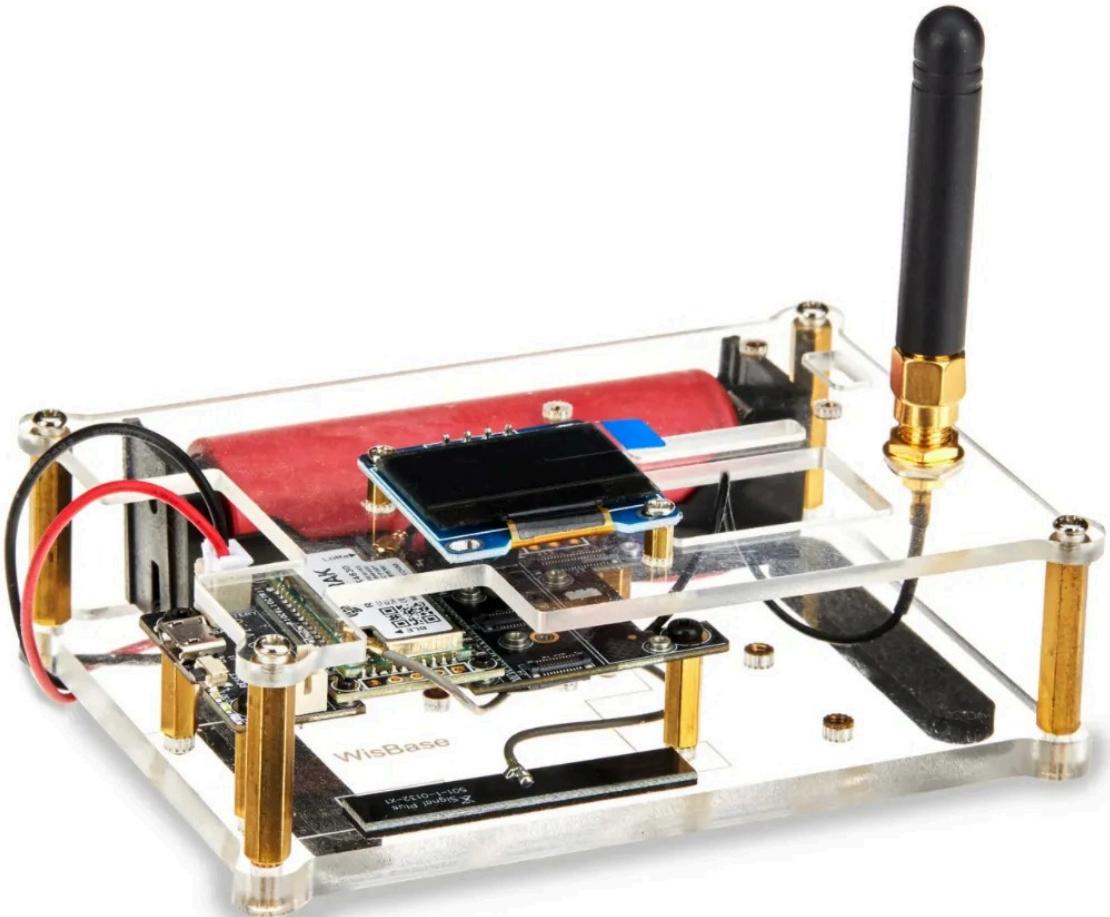
Screen Support

OLED screen support (OLED screen sold separately)

 **NOTE**

The RAK5005 (without the -O) is not compatible.

Further information on the RAK5005-O can be found on the RAK Documentation Center.



RAK19007

RAK19007 - WisBlock Base Board (2nd Generation, an upgrade to the RAK5005-O)

Slots

- (x1) Core Module slot
- (x1) WisBlock IO Module slot
- (x4) WisBlock Sensor Module slots

Buttons

- (x1) Reset Button

It may be possible to add a user button using the 13002 IO module.

Connectors

Connector for 3.7v LiPo battery (with charge controller)

Connector for 5v solar panel (max 5.5v)

I²C, UART, BOOT and GPIOs accessible with solder contacts

USB-C port for debugging and power

Screen Support

OLED screen support (OLED screen sold separately)

Further information on the RAK19007 can be found on the RAK Documentation Center.

Resources

Purchase Links:

US

Rokland

International

RAK Wireless

RAK Wireless Starter Kit

RAK19003

RAK19003 - WisBlock's Mini Base Board.

Slots

- (x1) Core Module slot
- (x2) WisBlock Sensor Module slots

Buttons

- (x1) Reset Button

There is a 3rd party sensor module that breaks out the user button, available here.

Connectors

Connector for 3.7v LiPo battery (with charge controller)

Connector for 5v solar panel (max 5.5v)

I²C, UART and BOOT headers accessible with solder contacts

USB-C port for debugging and power

Screen Support

OLED screen support (OLED screen sold separately)

Further information on the RAK19003 can be found on the RAK Documentation Center

Resources

Purchase Links:

US

Rokland

International

RAK Wireless



RAK19001

RAK19001 - WisBlock's Dual IO Base Board.

Slots

- (x1) Core Module slot
- (x2) WisBlock IO Module slot
- (x6) WisBlock Sensor Module slots

Buttons

- (x1) Reset Button
- (x1) User-defined push button switch
- (x1) Battery selector switch

On this board the PIN for user button (IO5) is available as a solder

contact on the upper header row.

Connectors

Connector for 3.7v LiPo battery (with charge controller)

Connector for 5v solar panel (max 5.5v)

Separate connector for non-rechargeable batteries

I²C, SPI, UART, BOOT and GPIOs accessible with solder contacts

USB-C port for debugging and power

Screen Support

OLED screen support (OLED screen sold separately)

Further information on the RAK19001 can be found on the RAK Documentation Center.

Resources

Purchase Links:

US

Rokland

International

RAK Wireless

RAK WisBlock Core Modules

RAK4631 - nRF52

ⓘ INFO

Please be aware of the difference between the RAK4631 (Arduino bootloader) and the RAK4631-R (RUI3 bootloader). Meshtastic requires the Arduino bootloader. If you have a RAK4631-R, please see the instructions for converting the bootloader.

RAK4631

MCU

nRF52840

Bluetooth BLE 5.0

Very low power consumption

LoRa Transceiver:

SX1262

Frequency Options:

433 MHz

470 MHz

799 MHz

865 MHz

868 MHz

915 MHz

920 MHz

923 MHz

Connectors:

U.FL/IPEX antenna connector for LoRa

Resources

Firmware file: [firmware-rak4631-X.X.X.xxxxxxx.uf2](#)

Further information on the RAK4631 can be found on the RAK Documentation Center.

Purchase Links:

US

Rokland - US915 Mhz

International

RAK Wireless Store

RAK Wireless Aliexpress



GPIO

NOTE

There is no usable GPIO pin on any RAK base board except the 'big' baseboard RAK19001 without adding a RAK13002 IO module or a third party IO sensor breakout.

The RAK4631 uses symbolic labels for its I/O Pins on the module and baseboard silk screens. The following table shows the mapping of the RAK4631 GPIO pins to the corresponding Arduino pins and the MCU Port numbers.

RAK Pin	nRF52840 Pin	Arduino GPIO	Remark
IO1	P0.17	17	used to toggle power to peripheral modules using 3v3_S power rail, not available for user application
IO2	P1.02	34	used to power all peripheral modules, not available for user application
IO3	P0.21	21	
IO4	P0.04	4	

RAK Pin	nRF52840 Pin	Arduino GPIO	Remark
IO5	P0.09	9	The 'User Button' is mapped here.
IO6	P0.10	10	
IO7	P0.28	28	
SW1	P0.01	1	
A0	P0.04/AIN2	A2	
A1	P0.31/AIN7	A7	
SPI_CS	P0.26	26	

When configuring GPIO pins in your device settings, the Arduino GPIO numbers should be used.

Example

```
meshtastic --set external_notification.output 10
```

This will use IO6 on a RAK4631

RAK11200 - ESP32



Only supported on the RAK5005-O base board.

The RAK11200 does not contain a LoRa transceiver, and thus needs to be added separately in the form of the RAK13300 LPWAN module. This occupies the IO Port of the base board.

RAK11200

MCU:

ESP32-WROVER

Bluetooth 4.2

WiFi 802.11 b/g/n

High power consumption (relative to nRF52)

Resources

Firmware file: [firmware-rak11200-X.X.X.xxxxxxx.bin](#)

Further information on the RAK11200 can be found on the RAK Documentation Center.

Purchase Links:

US

Rokland

International

[RAK Wireless Store](#)

[RAK Wireless Aliexpress](#)



RAK11310 - RP2040

ⓘ INFO

Please note, this core module does NOT include BLE/
WiFi.

MCU:

Raspberry Pi RP2040

Dual M0+ Core

133MHz CPU Clock

LoRa Transceiver:

SX1262

Frequency Options:

433 MHz

470 MHz

864 MHz

865 MHz

868 MHz

915 MHz

920 MHz

923 MHz

Connectors:

U.FL/IPEX antenna connector for LoRa

Resources

Firmware file: [firmware-rak11310-X.X.X.xxxxxxx.uf2](#)

Further information on the RAK11310 can be found on the RAK Documentation Center.

Purchase Links:

US

Rokland

International

RAK Wireless Store

RAK Wireless Aliexpress

RAK WisBlock Screens

There are currently two different screens supported by the RAK WisBlock system:

RAK1921 OLED Display

The RAK1921 OLED display is a 0.96 inch monochrome display.

0.96 inch OLED display

Resolution 128 x 64 pixels

I²C interface

This item requires soldering. Similar modules are widely available from other suppliers, but do check the boards as some have the VDD and GND pins swapped round. This will prevent directly soldering the display to the baseboard. The preferred order is VDD, GND, SCL, SDA. If pin ordering on the OLED board are swapped, there are some tricks to allow either reconfiguring the pins of the OLED via soldered jumpers, or by carefully soldering wire for those pins that are out-of-sequence. The final option is to use longer wires between the board and display, which permits re-ordering the wires as required.

Resources

RAK Documentation Center

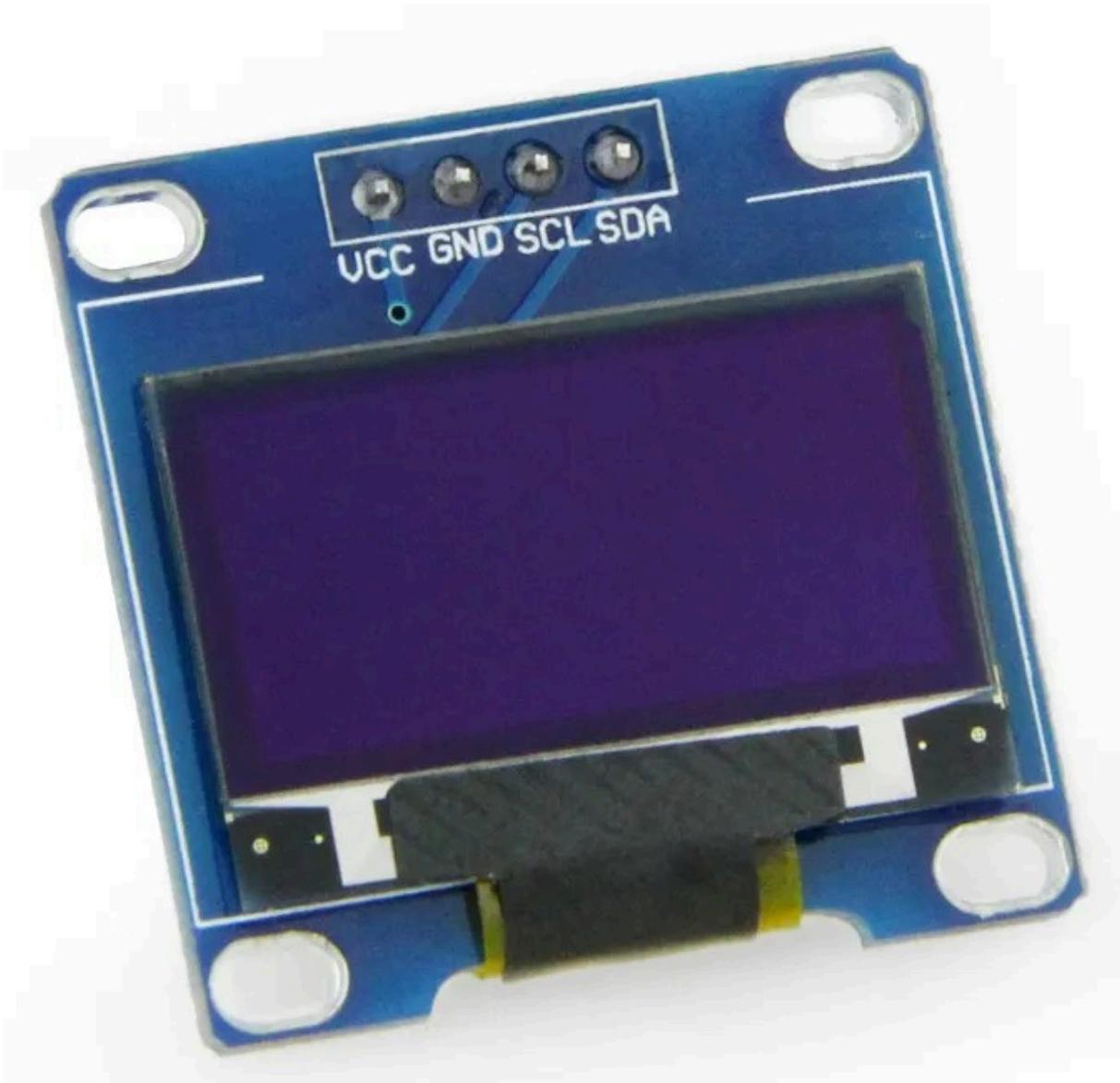
Purchase Links:

US

Rokland

International

RAK Wireless



RAK1400 E-Ink Display

The RAK1400 EPD module is an ultra low power E-Ink display with three user buttons.

2.13 inch black and white E-Ink display

Three button module

Resolution 212 x 104 pixels

Occupies the IO Port of a Wisblock Base

Please note only the white-black display is supported at this time,

the white-black-red display may work, but is not supported.

Resources

Firmware for 5005 with RAK14000 e-paper: [firmware-rak4631_eink-X.X.X.xxxxxxx.uf2](#)

RAK Documentation Center

Purchase Links:

US

Rokland

International

RAK Wireless



RAK WisBlock Hardware Buttons

Functionality

Button functionality for RAK devices greatly depends on the device specific configuration. If your device has any of the following buttons, the functionality is generally the same for all RAK devices:

RAK5005-O / RAK19007 / RAK19003

Reset Button:

Single press: Resets the device.

Double press: Put device into bootloader mode.

User/Program(if configured):

Long press: Will signal the device to shutdown after 5 seconds.

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

RAK19001

Reset Button:

Single press: Resets the device.

Double press: Put device into bootloader mode.

User/Program Button (if configured):

Long press: Will signal the device to shutdown after 5 seconds.

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

RAK WisBlock Supported Peripherals

GPS Modules

RAK12500

The RAK12500 GPS sensor is a newer GPS module and is generally preferred.

uBlox Zoe-M8Q GNSS receiver

GPS, GLONASS, QZSS and BeiDou satellite support

The RAK12500 is supported on the following base boards & slots:

RAK19007 on slot A

RAK19003 on slot C

RAK1910

The RAK1910 GPS sensor is the older of the supported GPS modules for RAK boards.

uBlox MAX-7Q GPS module

GPS and GLONASS satellite support

The RAK1910 is supported on the following base boards & slots:

RAK5005-0 on slot A

RAK19007 on slot A

RAK19003 on slot C

Resources

[RAK Documentation Center](#)

[RAK12500](#)

RAK1910

Purchase Links:

US

Rokland RAK1910

Rokland RAK12500

International

RAK Wireless RAK1910

RAK Wireless RAK12500

Buzzer

The RAK18001 Buzzer Module is currently being tested for integration with the External Notifications plugin. There is currently a known conflict with buzzer if the module is placed in Slot D, although other slots should work.

Resources

RAK Documentation Center RAK18001

Purchase Links:

International

RAK Wireless

IO Module

The RAK13002 IO Module can be used to, among other things, add a user button to the RAK base boards (excluding the RAK19003 Mini base board). It features a number of different interface options:

2x I²C interfaces

2x UART interfaces

1x SPI interface

Up to 6x GPIOs

2x ADC interfaces

3.3v Power rails

There is development activity in progress to get sensors such as this added to the Meshtastic Core.

Resources

RAK Documentation Center RAK13002

Purchase Links:

US

Rokland

International

RAK Wireless

Environmental Sensors

RAK1901 Temperature and Humidity Sensor

The RAK1901 Temperature and Humidity Sensor is based on the Sensirion SHTC3 module and has the following features:

Temperature measurement (Range -40°C to +125°C)

Humidity measurement (Range 0% to 100%)

Lower power consumption

RAK-1902 Barometric Pressure Sensor

The RAK1902 Barometric Pressure Sensor is based on the STMicroelectronics LPS22HB module and has the following features:

Barometer measurement (Range 260 to 1260 hPa)

Low power consumption of 3uA

Small form factor

RAK1906 Environment Sensor

The RAK1906 Environment Sensor is based on the Bosch BME680 module and has the following features:

Temperature measurement (Range -40°C to +85°C)

Humidity measurement (Range 0% to 100%)

Barometer measurement (Range 300 to 1100 hPa)

Air Quality measurement

Resources

RAK Documentation Center

RAK1901

RAK1902

RAK1906

Purchase Links:

US

Rokland RAK1901

Rokland RAK1902

Rokland RAK1906

International

RAK Wireless RAK1901

RAK Wireless RAK1902

RAK Wireless RAK1906

LILYGO® TTGO T-Beam Devices

All T-beam models (with the exception of the S3-Core) have an 18650 size battery holder on the rear of the device. This is designed to the original specification of the 18650 and only fits unprotected flat top 18650 cells. Button top and protected cells are typically longer than 65mm, often approaching 70mm.

Further information on the LILYGO® T-Beam devices can be found on LILYGO®'s GitHub page.

T-Beam v0.7

FIRMWARE VERSION NOTICE

This is an earlier version of the T-Beam board. Due to changes in the design this board uses a specific firmware file different from the other T-Beam boards. This board is no longer in production and not recommended for new purchases.

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1276

Frequency options

433 MHz

868 MHz

915 MHz

Navigation Module

NEO-6M (GPS receiver)

Connectors

Micro USB

Antenna: SMA antenna connector

Features

Meshtastic preinstalled

Power, Program and Reset switches

Screen sold separately

No GPS

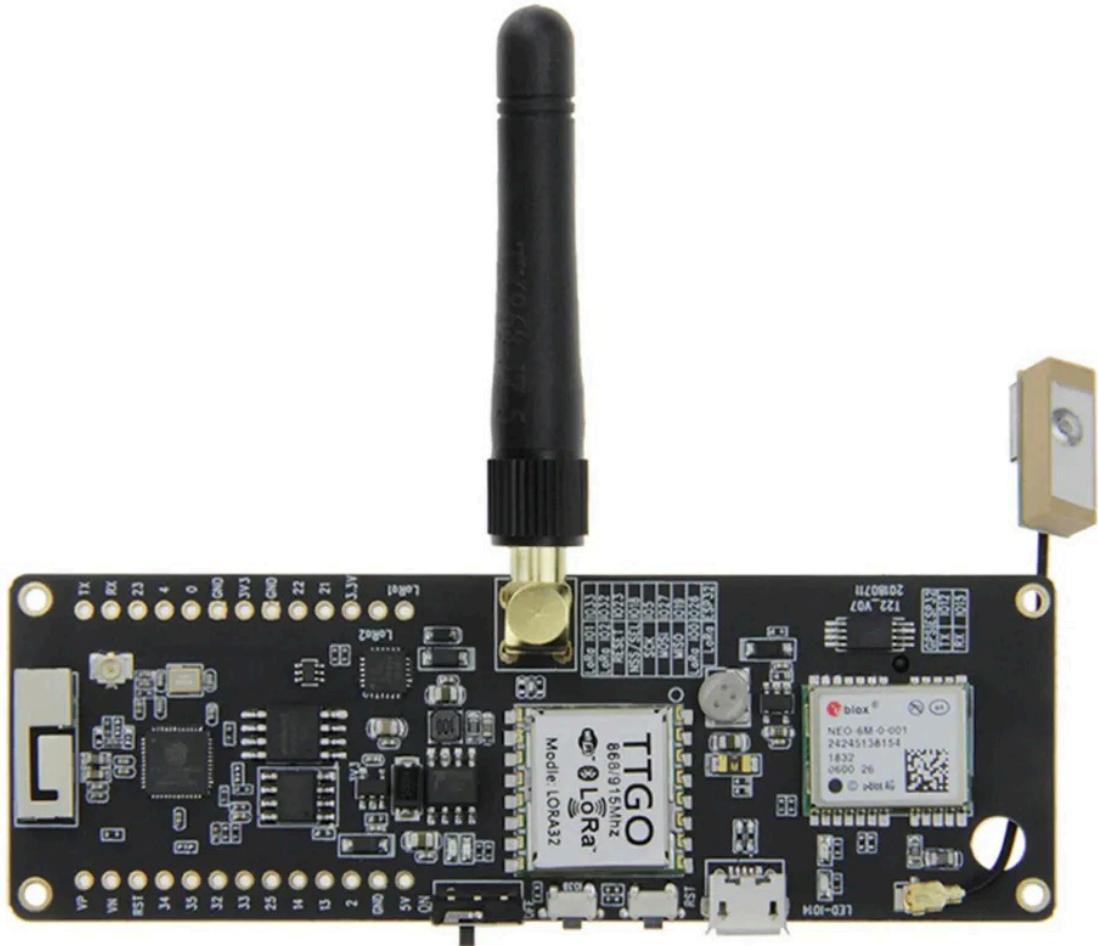
Resources

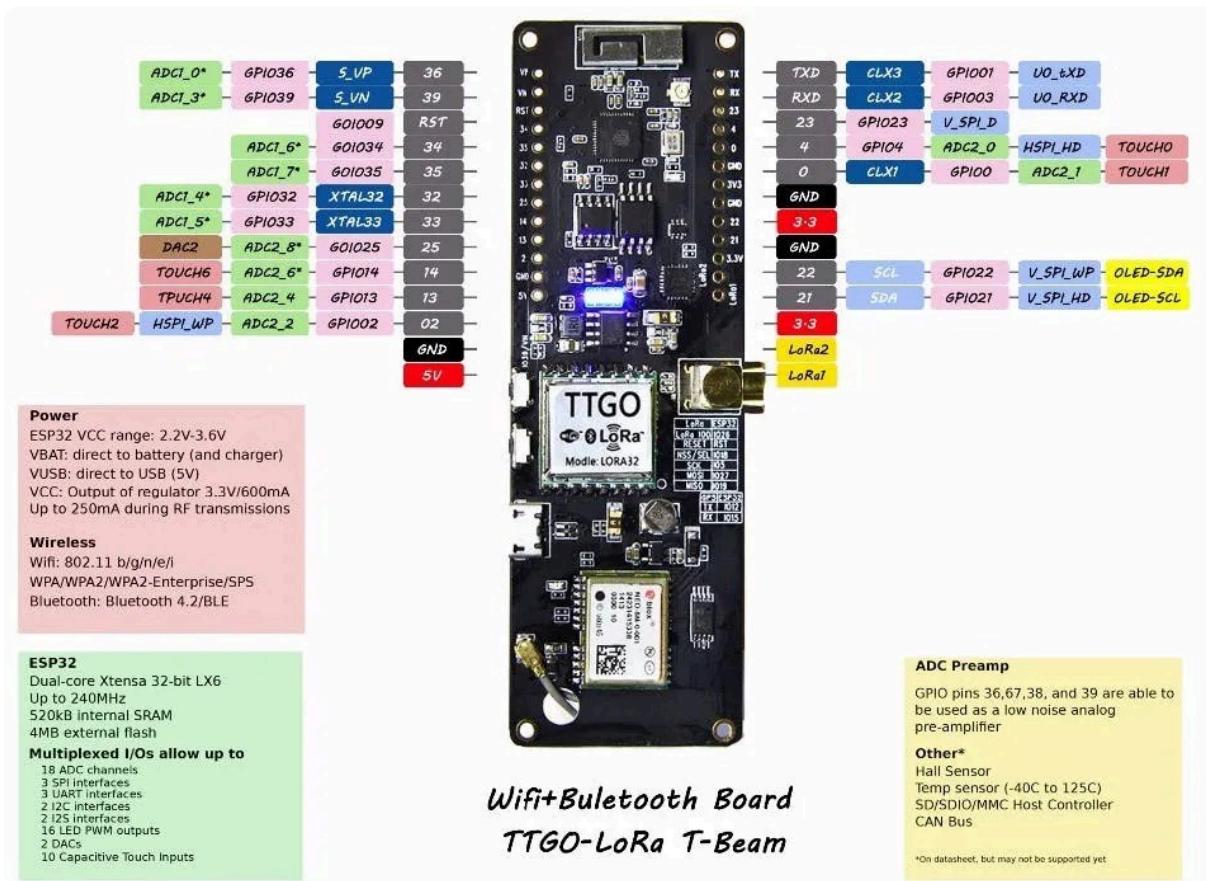
Firmware file: [firmware-tbeam0.7-X.X.X.xxxxxxx.bin](#)

Purchase Links:

International

AliExpress





T-Beam v1.1

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1276

Frequency options

433 MHz

868 MHz

915 MHz

923 MHz

Navigation Module

NEO-6M (GPS receiver)

Connectors

Micro USB

Antenna: SMA antenna connector

Features

Meshtastic preinstalled

Power, Program and Reset switches

Comes with 0.96 inch OLED display (soldering required to assemble)

Resources

Firmware file: [firmware-tbeam-X.X.X.xxxxxxx.bin](#)

Purchase Links:

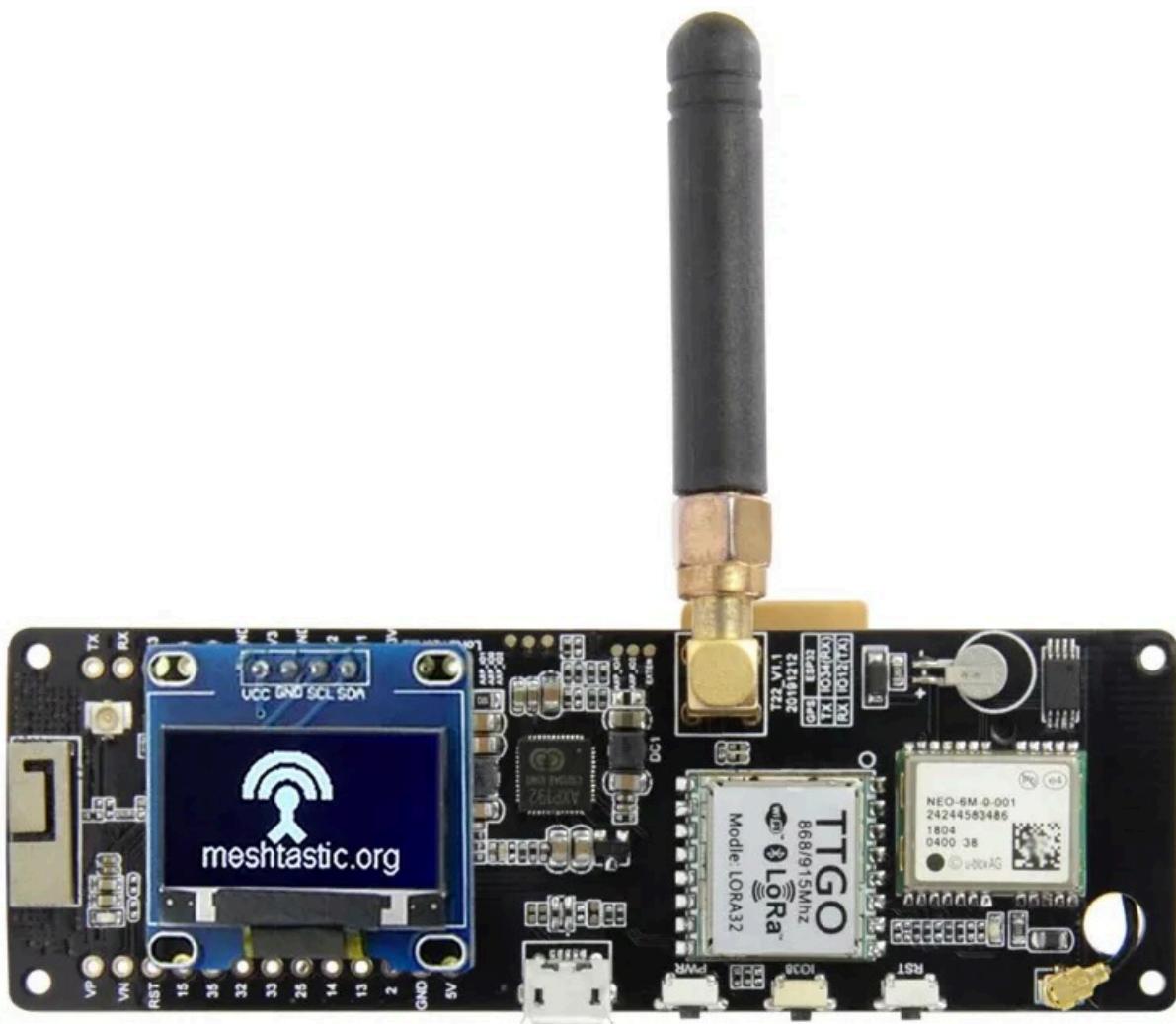
US

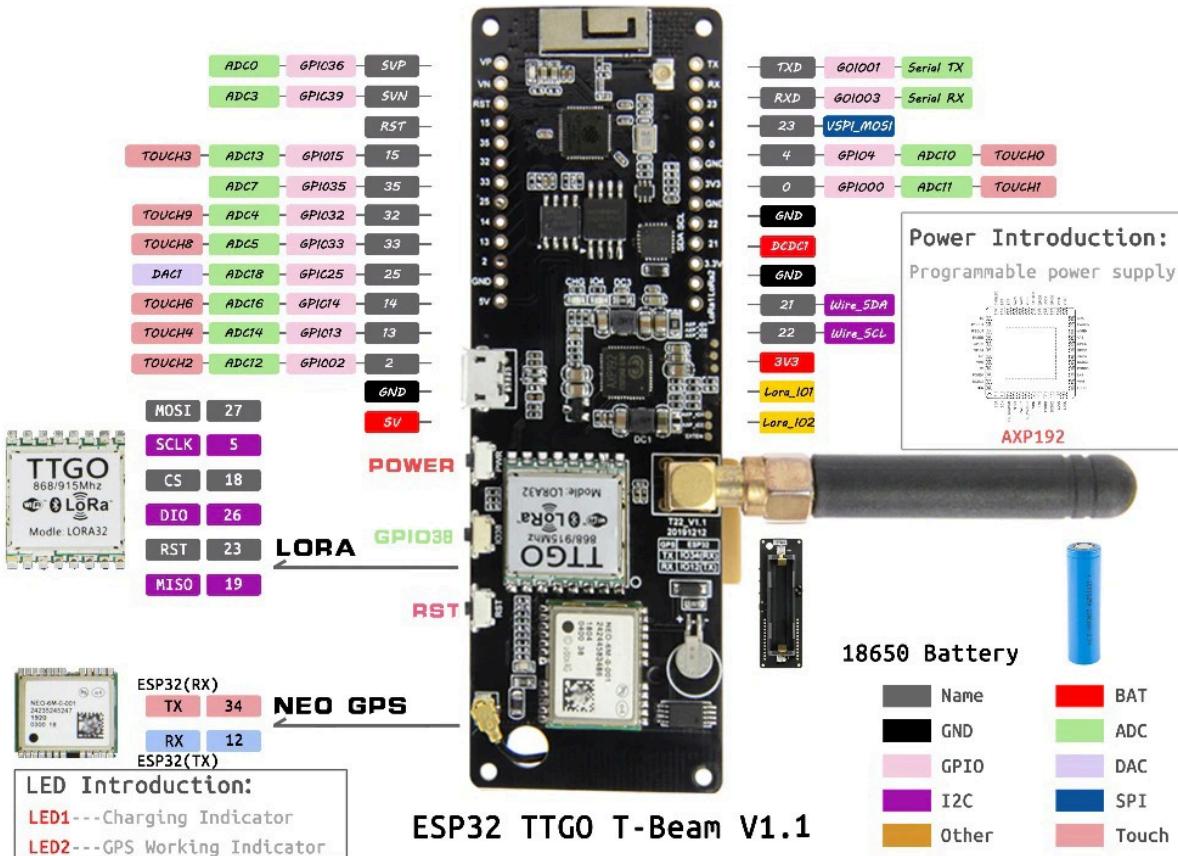
Rokland OLED

Rokland No OLED

International

AliExpress





T-Beam with M8N and SX1276 MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1276

Frequency options

433 MHz

868 MHz

915 MHz

923 MHz

Navigation Module

NEO-M8N - GNSS receiver (supports GPS, GLONASS, Galileo, BeiDou) (better GPS sensitivity)

Connectors

Micro USB

Antenna: U.FL antenna connector

Features

Meshtastic preinstalled

Power, Program and Reset switches

Screen sold separately

Resources

Firmware file: [firmware-tbeam-X.X.X.xxxxxxx.bin](#)

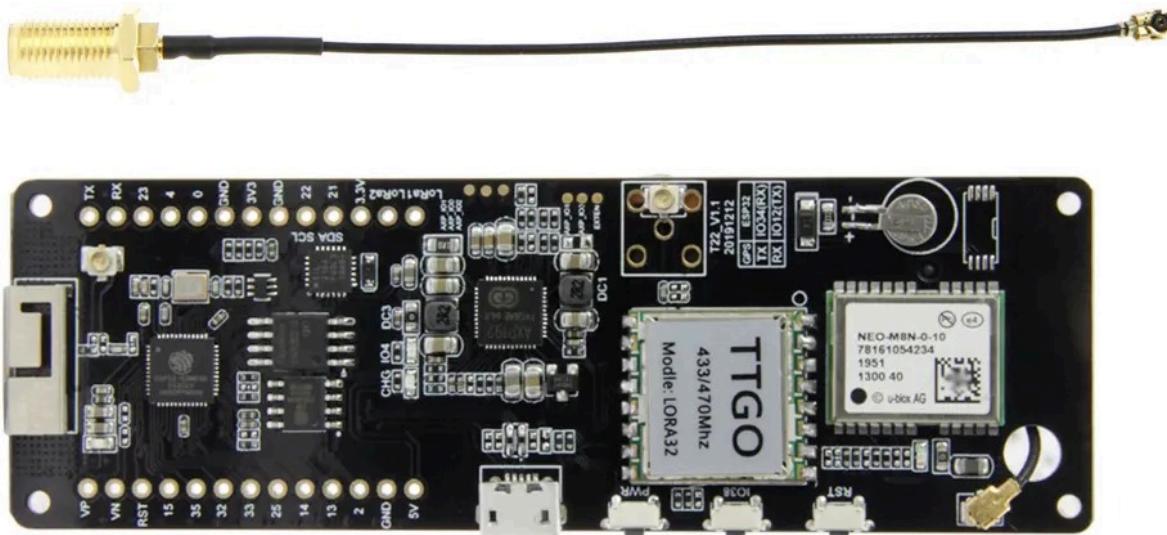
Purchase Links:

US

Rokland

International

AliExpress



T-Beam with M8N and SX1276 MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1262 (improved performance)

Frequency options

433 MHz

868 MHz

915 MHz

923 MHz

Navigation Module

NEO-M8N - GNSS receiver (supports GPS, GLONASS, Galileo, BeiDou) (better GPS sensitivity)

Connectors

Micro USB

Antenna: U.FL antenna connector

Features

Meshtastic preinstalled

Power, Program and Reset switches

Screen sold separately

Resources

Firmware file: [firmware-tbeam-X.X.X.xxxxxxx.bin](#)

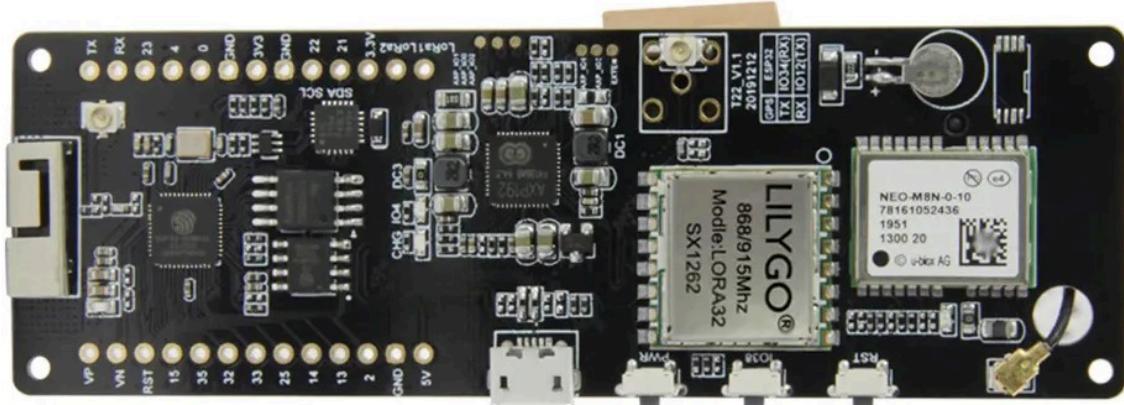
Purchase Links:

US

Rokland

International

AliExpress



T-Beam S3 Core

MCU

ESP32-S3 (WiFi & Bluetooth 5LE)

LoRa Transceiver

Semtech SX1262 (improved performance)

Frequency options

433 MHz

868 MHz

915 MHz

Navigation Module

NEO-M10S - GNSS receiver (supports GPS, GLONASS,

Galileo, BeiDou) (better GPS sensitivity)

Quectel L76K - (supports GPS, Beidou, GLONASS) (lower

price)

Connectors

USB-C

Antenna: U.FL antenna connector

Features

SoftRF preinstalled (flashing to Meshtastic required)

Boot and Reset switches

Can be used standalone without 'Supreme' daughterboard in a
headless configuration

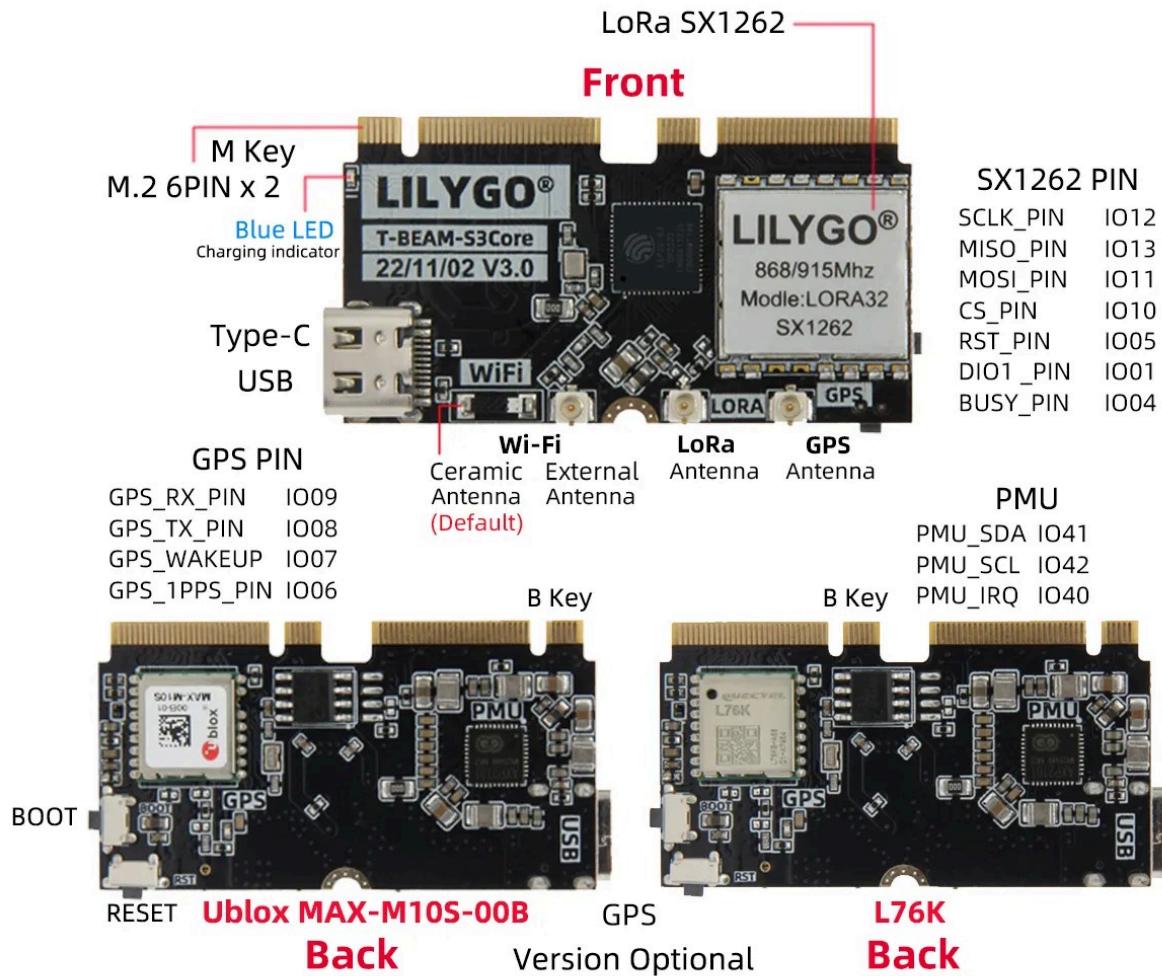
Resources

Firmware file: [firmware-tbeam-s3-core-X.X.X.xxxxxxx.bin](#)

Purchase Links:

International

AliExpress



LILYGO T-BEAM-S3Core PINMAP

LILYGO ESP32-S3 + LoRa + GPS + PMU + Antenna x 3

T-Beam Supreme

MCU

ESP32-S3 (WiFi & Bluetooth 5LE)

LoRa Transceiver

Semtech SX1262 (improved performance)

Frequency options

433 MHz

868 MHz

915 MHz

Navigation Module

NEO-M10S - GNSS receiver (supports GPS, GLONASS,

Galileo, BeiDou) (better GPS sensitivity)

Quectel L76K - (supports GPS, Beidou, GLONASS) (lower price)

Connectors

USB-C

Antenna: U.FL antenna connector

Features

Includes T-Beam S3-Core Module

SoftRF preinstalled (flashing to Meshtastic required)

Power, Boot and Reset switches

1.3" OLED included

BME280 Air pressure sensor

QMI8658 IMU

QMC6310 Magnetometer

PCF8563 RTC

Micro-SD reader (not implemented in Meshtastic)

Flashing

If you are having issues flashing the T-Beam Supreme, especially if it's your first attempt, you may need to manually place the device into Espressif's Firmware Download mode. Please follow the process below to do so.

⚠️ WARNING

Do not proceed unless an antenna is connected to avoid possible damage to the device's radio.

The following process will manually place the device into the

Espressif Firmware Download mode:

Unplug the device.

Press and hold the BOOT button.

Plug device in.

After 2-3 seconds, release the BOOT button.

With the device now in the Espressif Firmware Download mode, you can proceed with flashing using one of the supported flashing methods. It's generally recommended to use the Web Flasher. You can select "Tbeam S3 Core from the device drop-down.

Resources

Firmware file: `firmware-tbeam-s3-core-X.X.X.xxxxxxx.bin`

Purchase Links:

International

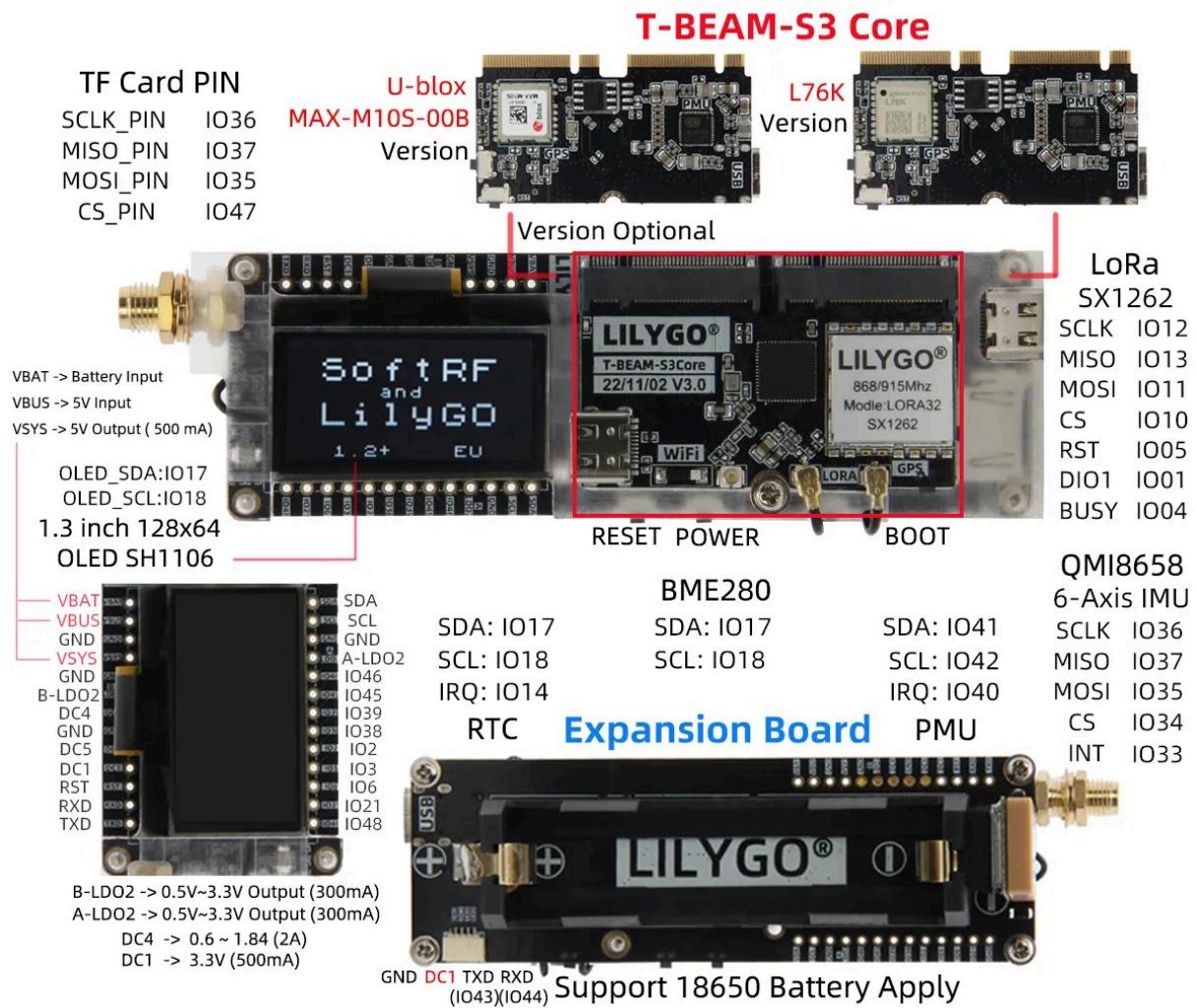
LilyGO Store

AliExpress

US

Rokland NEO-M10S

Rokland Quectel L76K



LILYGO SoftRF T-BEAM-S3 SUPREME

ESP32-S3 + LoRa + GPS +OLED + IMU + PMU + TF Card

T-Beam Hardware Buttons

Functionality

Power Button (left):

Long press: Powers the device on or off.

Reset Button (right):

Single press: Resets the device.

User/Program Button (middle):

Long press: Will signal the device to shutdown after 5 seconds.

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

Triple press: Disables the device's GPS. Repeat to re-enable.

(This will be indicated on both information screen pages on the device's display as shown below)



T-Beam Screens

0.96 inch OLED I²C display

Pin map

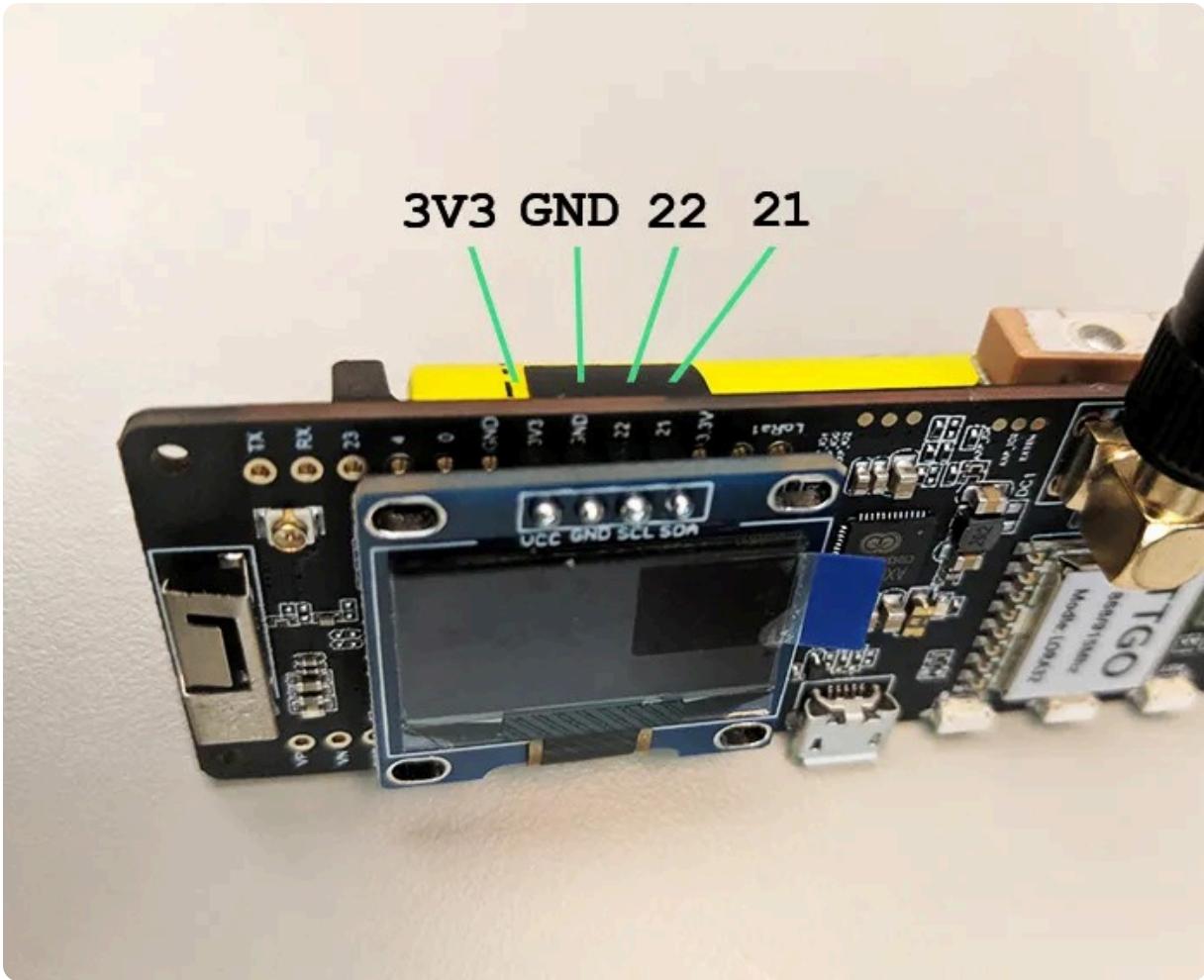
To attach the screen:

Connect VCC to 3v3

Connect GND to GND

Connect SCL to pin 22

Connect SDA to pin 21



Resources

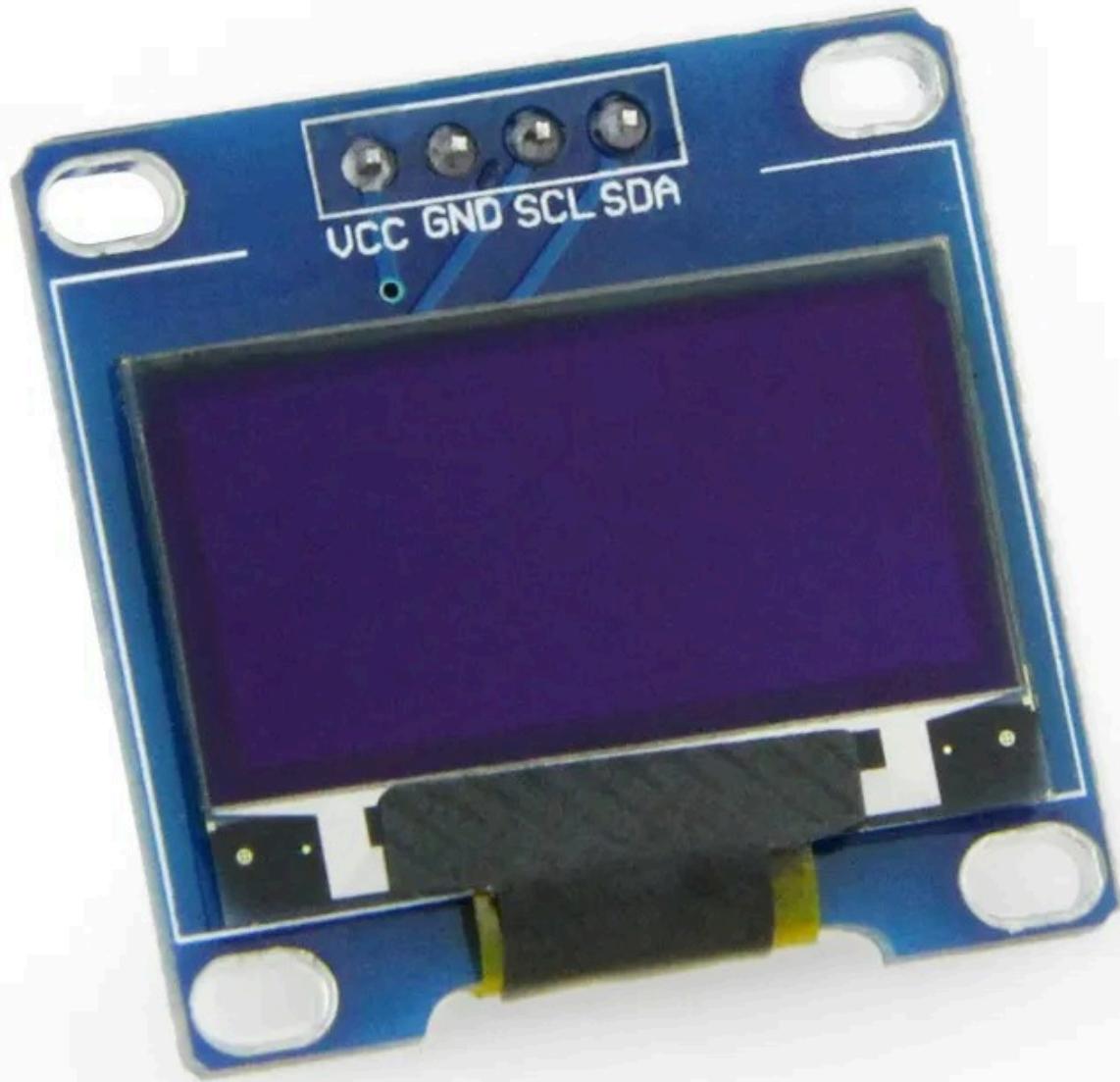
Purchase Links:

US

Rokland

International

Aliexpress



LILYGO® TTGO T-Echo devices

The T-Echo is the latest device to be release by LILYGO® supporting a low power consumption micro-controller.

Further information on the LILYGO® T-Echo devices can be found on LILYGO®'s GitHub page.

Specifications

MCU

nRF52840 (Bluetooth BLE 5.0 & NFC)

LoRa Transceiver

Semtech SX1262

Frequency options

433 MHz

868 MHz

915 MHz

Navigation Module

L76K GNSS receiver (Supports GPS, BeiDou, GLONASS & QZSS)

Connectors

U.FL antenna connector

Features

Reset, Program and capacitive touch buttons
1.54" eInk display
Optional BME280 - Humidity and Pressure Sensor
Comes with a case and battery

Resources

Firmware file: [firmware-t-echo-2.x.x.uf2](#)

Purchase Links:

US

Rokland 915MHz

Rokland 915MHz w/ BME280 Kit

International

LilyGO

AliExpress



T-Echo Hardware Buttons

Functionality

Capacitive Touch Button (Top):

Short press: Updates the e-ink display.

Reset Button (Button 1):

Single press: Resets the device.

Double press: Puts the device into bootloader mode which allows you to update the firmware.

Program/Power Button (Button 2):

Single press: Changes the information page displayed on the device's screen.

Double press: Turns the screen backlight on/off and sends an adhoc ping of the device's position to the network.

Long press: Signals the device to shutdown after 5 seconds.



LILYGO® TTGO Lora Devices

Further information on the LILYGO® LoRa devices can be found on LILYGO®'s GitHub page.

Lora v1

WARNING

Not recommended with a battery! These boards contain the wrong component in the LiPo battery charging circuit allowing the battery to be overcharged.

CAUTION

This board is still in production but for various reasons not recommended for new purchases or for unattended installations. Firmware support is phased out. If in doubt, choose the Lora V2.1-1.6 or Lora T3S3 board.

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1276

Frequency options

915 MHz

868 MHz

Connectors

Micro USB

Antenna: U.FL antenna connector

Features

Built in 0.96 inch OLED display

Resources

Firmware file: [firmware-tlora-v1-X.X.X.xxxxxxx.bin](#)

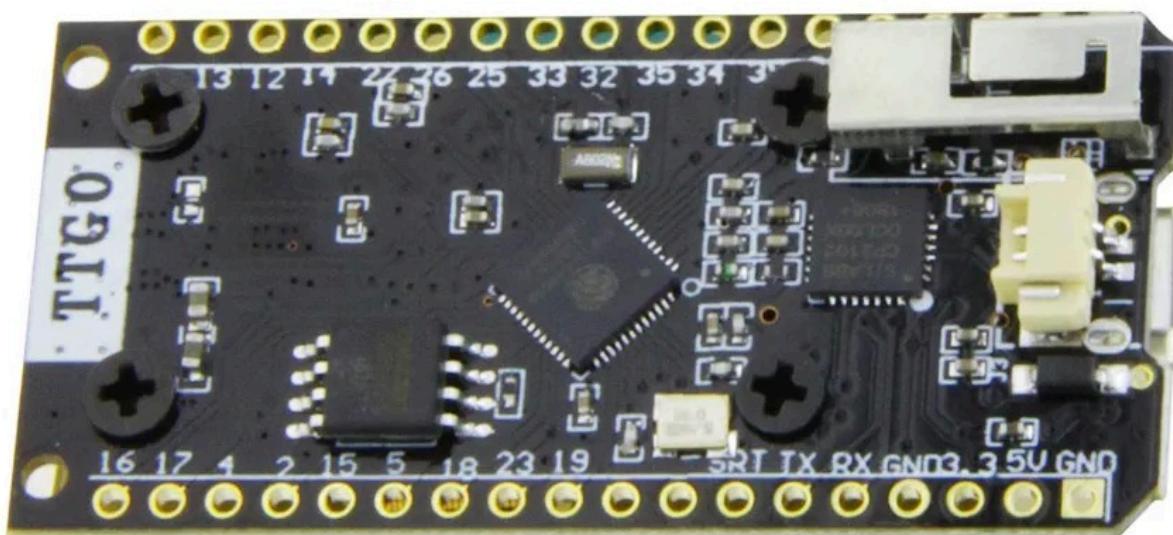
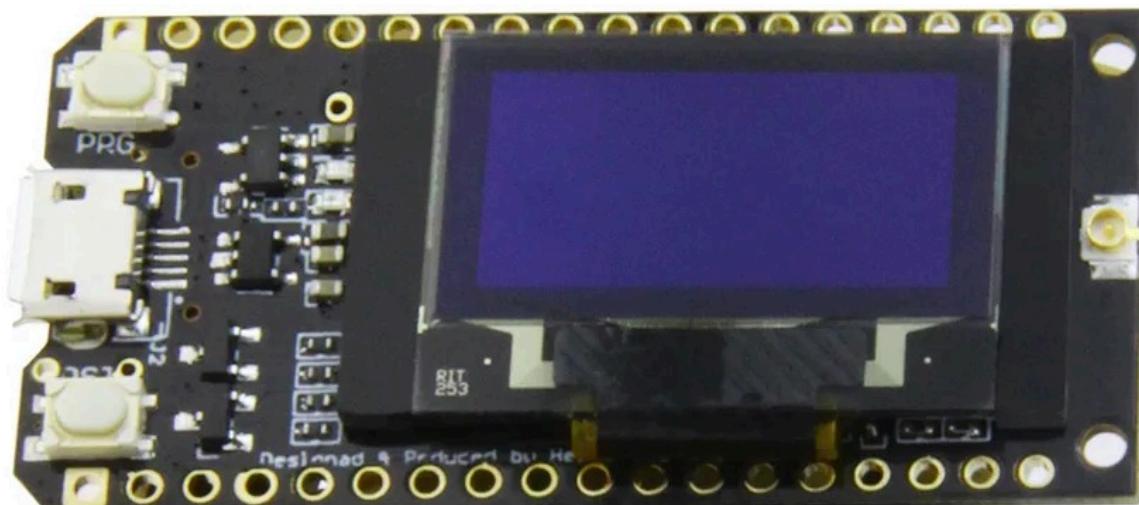
Purchase Links:

US

Rokland

International

AliExpress



Lora v1.3

⚠️ WARNING

Not recommended with a battery! These boards contain the wrong component in the LiPo battery charging circuit allowing the battery to be overcharged.

⚠️ CAUTION

This board is still in production but for various reasons not recommended for new purchases or for unattended installations. Firmware support is phased out. If in doubt, choose the Lora V2.1-1.6 or Lora T3S3 board.

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX127x

Frequency options

915 MHz

868 MHz

Connectors

Micro USB

Antenna: U.FL antenna connector

Features

Built in 0.96 inch OLED display

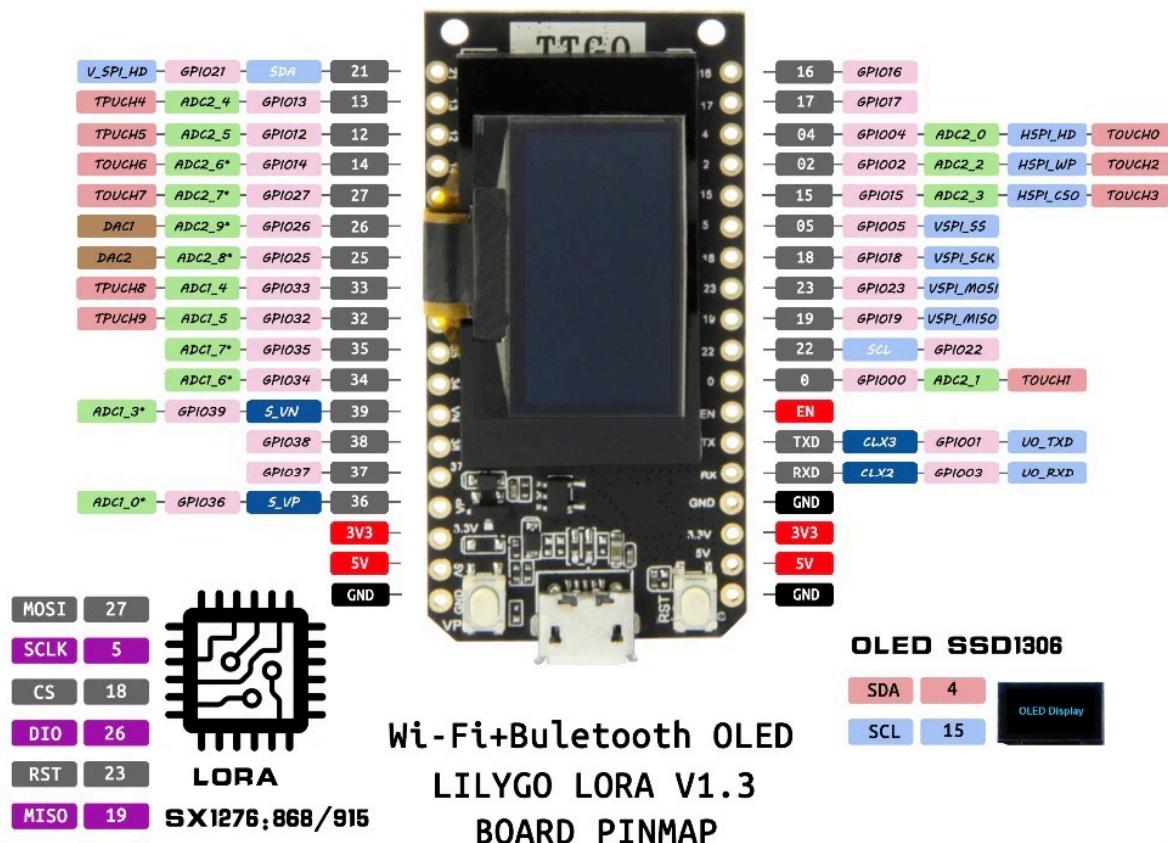
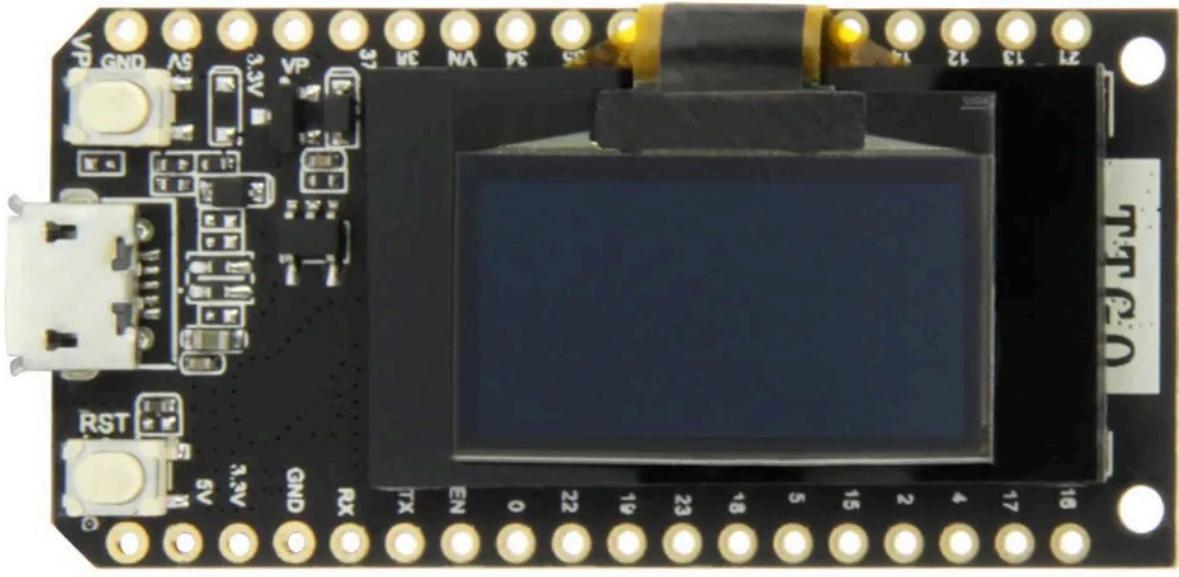
Resources

Firmware file: [firmware-tlora_v1_3-X.X.X.xxxxxxx.bin](#)

Purchase Links:

International

AliExpress



Lora V2.0

WARNING

Not recommended with a battery! These boards contain the wrong component in the LiPo battery charging circuit allowing the battery to be overcharged.

CAUTION

This board is still in production but for various reasons not recommended for new purchases or for unattended installations. Firmware support is phased out. If in doubt, choose the Lora V2.1-1.6 or Lora T3S3 board.

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX127x

Frequency options

433 MHz

868 MHz

915 MHz

Connectors

Micro USB

Antenna: U.FL antenna connector

Features

Built in 0.96 inch OLED display

Power and Reset switches

microSD connector

No GPS

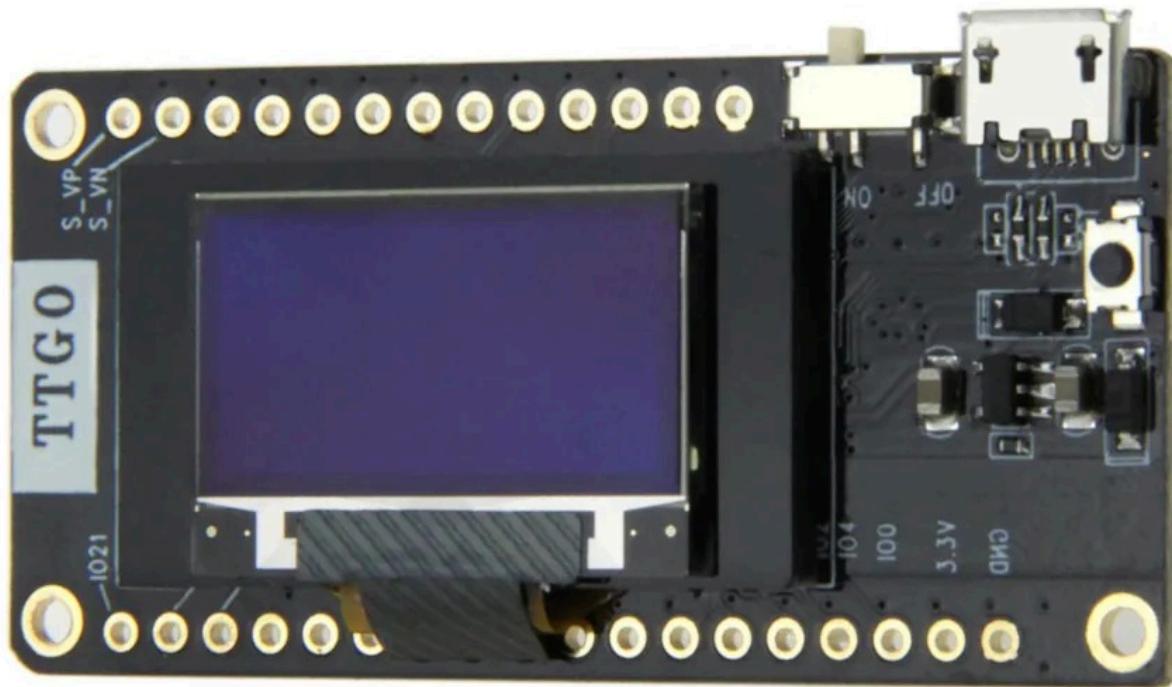
Resources

Firmware file: [firmware-tlora-v2-X.X.X.xxxxxxx.bin](#)

Purchase Links:

International

AliExpress



Lora v2.1-1.6

⚠ CAUTION

Early versions of some of these boards contained the wrong component in the LiPo battery charging circuit allowing the battery to be overcharged. Boards purchased after 2021

should be ok.

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX127x

Frequency options

433 MHz

868 MHz

915 MHz

Connectors

Micro USB

Antenna: SMA antenna connector

Variations of identifying marks

FCC ID: 2ASYE-T3-V1-6-1

"T3_v1.6.1 20210104" on the board

"T3_v1.6 20180606" on the board

"Model T3 V1.6.1" on the FCC sticker

Features

Built in 0.96 inch OLED display

Power and Reset switches

microSD connector

No GPS

Resources

Firmware file: [firmware-tlora-v2-1-1.6-X.X.X.xxxxxxx.bin](#)

Purchase Links:

US

Rokland

International

AliExpress

Lilygo



Lora v2.1-1.8

MCU

ESP32 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1280 (Region LORA_24 worldwide use)

Frequency options

2.4 GHz

Connectors

USB-C

Antenna: SMA antenna connector

Features

Built in 0.96 inch OLED display

Power and Reset switches

microSD connector

No GPS

Resources

Firmware file: [firmware-tlora-v2-1-1.8-X.X.X.xxxxxxx.bin](#)



Lora T3S3 v1.0

MCU

ESP32-S3 (WiFi & Bluetooth)

LoRa Transceiver

Semtech SX1262

Semtech 1276

Semtech SX1280 with PA (Region LORA_24 worldwide use)

Frequency options

868 MHz

915 MHz

2.4 GHz

Connectors

USB-C

Antenna: SMA antenna connector

Features

Built in 0.96 inch OLED display

Power and Reset switches, Boot / User Button

microSD connector

No GPS

Flashing the T3S3

⚠️ WARNING

Do not proceed unless an antenna is connected to avoid possible damage to the device's radio.

The following process will manually place the device into the Espressif Firmware Download mode:

Switch off the device.

Connect the USB-C data cable to the device. A blue LED will illuminate while display stays black.

Press and hold the BOOT button to the right of the display.

Switch the device on.

After 2-3 seconds, release the BOOT button.

With the device now in the Espressif Firmware Download mode,

you can proceed with flashing using one of the supported flashing methods. It's generally recommended to use the Web Flasher. You can select "Tlora T3s3 V1" from the device drop-down.

 **NOTE**

If after successfully flashing the device and the screen remains black, you may need to use the CLI Script to flash Meshtastic.

Resources

Firmware file: [firmware-tlora-t3s3-v1.xxxxxxx.bin](#)

Purchase Links:

US

Amazon

International

AliExpress

Lilygo

Tindie



LILYGO® TTGO Lora Hardware Buttons

Functionality

Reset Button:

Single press: Resets the device.

User/Program Button:

Long press: Will signal the device to shutdown after 5 seconds.

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

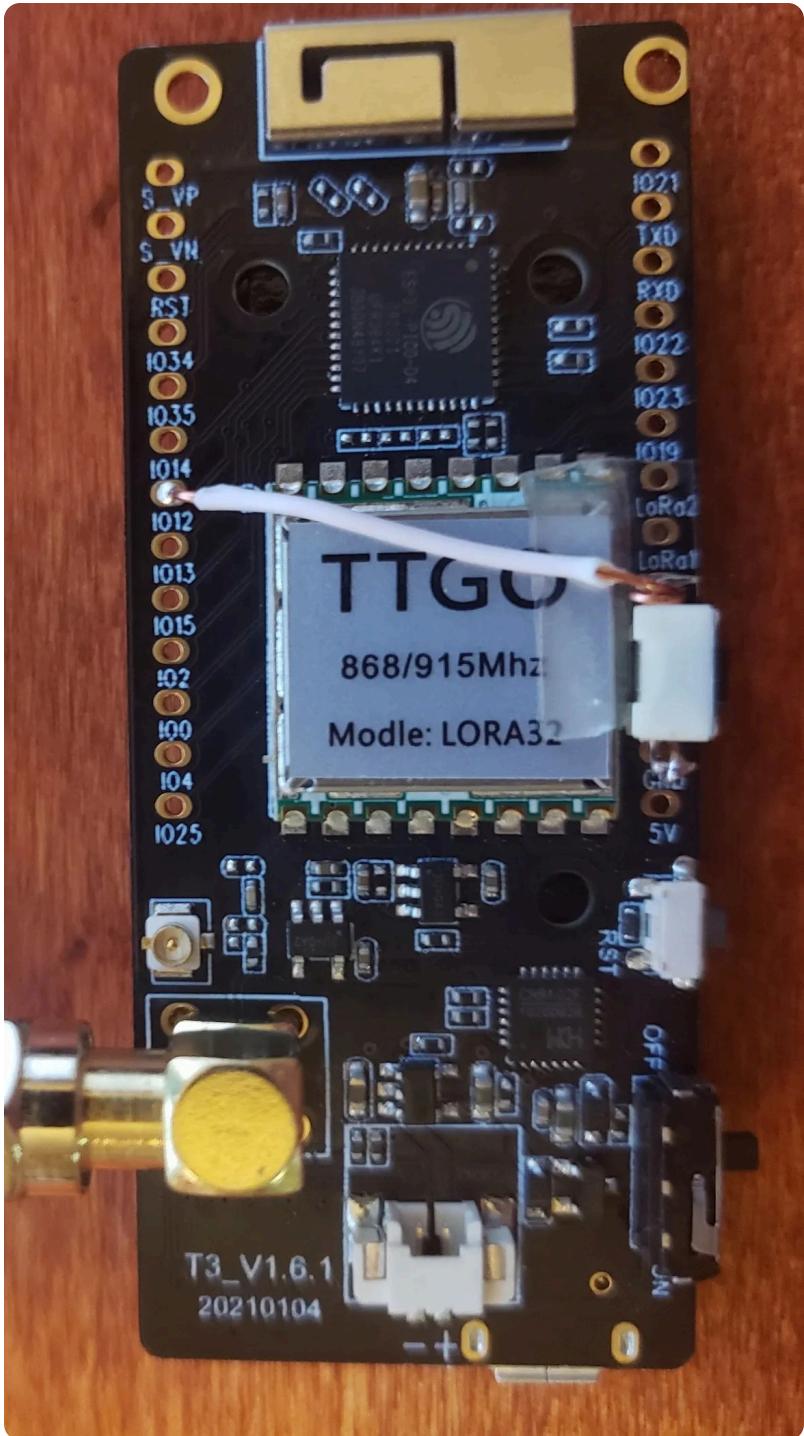
LILYGO® TTGO Lora

GPIO

GPIO IO12

For the Lora V2.1-1.6 and V2.1-1.8, Shorting IO12 to ground will progress through the screen pages and/or wake up the device. A simple push switch can be added for this purpose.





LILYGO® T-Deck

The T-Deck is a compact device featuring a 2.8-inch IPS LCD touch screen with a resolution of 320x240 pixels, integrated with a small keyboard, trackball, microphone and speaker running on an ESP32-S3 dual-core processor.

Specifications

MCU

ESP32-S3FN16R8 (WiFi & Bluetooth 5 LE)

LoRa Transceiver

Semtech SX1262

Frequency options

915 MHz

868 MHz

433 MHz

Antenna

U.FL/IPEX antenna connector for LoRa

Connectors

USB-C

Features

LILYGO® backlit T-Keyboard

Trackball

2.8 inch ST7789 SPI Interface IPS LCD (Resolution: 320 x 240)

I2S Speaker/Microphone

Keyboard Shortcuts

Shortcut	Function
<code>alt + b</code>	Toggle keyboard backlight on/off.

Flashing

If you are having issues flashing the T-Deck, especially if it's your first attempt, you may need to manually place the device into Espressif's Firmware Download mode. Please follow the process below to do so.

WARNING

Do not proceed unless an antenna is connected to avoid possible damage to the device's radio.

The following process will manually place the device into the Espressif Firmware Download mode:

Ensure the device's power switch is toggled OFF.

Press and hold the TRACKBALL.

Toggle device's power switch ON.

After 2-3 seconds, release the TRACKBALL button.

If the device screen is black and the backlight is off, the device is in the Firmware Download mode. If the backlight is on, repeat these steps.

With the device now in the Espressif Firmware Download mode, you can proceed with flashing using one of the supported flashing methods. It's generally recommended to use the Web Flasher. You can select "T Deck" from the device drop-down.

Resources

Firmware file: [firmware-t-deck-X.X.X.xxxxxxx.bin](#)

Purchase Links:

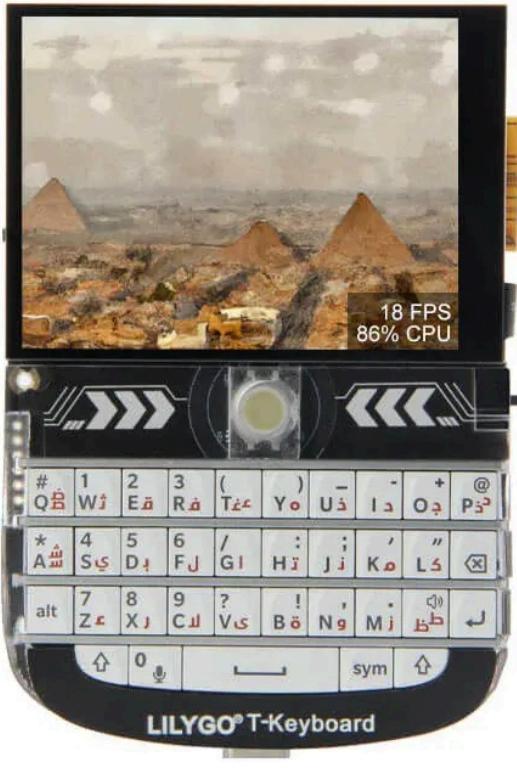
US

Rokland

International

LilyGO Store

AliExpress



LILYGO® T-Watch

The T-Watch S3 is a compact wearable device featuring a 1.54-inch IPS LCD touch screen with a resolution of 240x240 pixels. It includes haptic feedback, an integrated microphone, speaker, real-time clock, and a three-axis accelerometer.

Specifications

MCU

ESP32-S3 (WiFi & Bluetooth 5 LE)

LoRa Transceiver

Semtech SX1262

Frequency options

915 MHz

868 MHz

433 MHz

Antenna

U.FL/IPEX antenna connector for LoRa

Connectors

Micro-USB

Features

1.54 inch ST7789V TFT LCD (Resolution: 240 x 240)

DRV2605 Haptic Driver
I2S Speaker/Microphone & MAX98357 I2S Amp
AXP2101 PMU and 400mAh battery.
BMA423 Three Axis accelerometer
Real-Time Clock

Flashing

If you are having issues flashing the T-Watch S3, especially if it's your first attempt, you may need to manually place the device into Espressif's Firmware Download mode. Please follow the process below to do so.

 **WARNING**

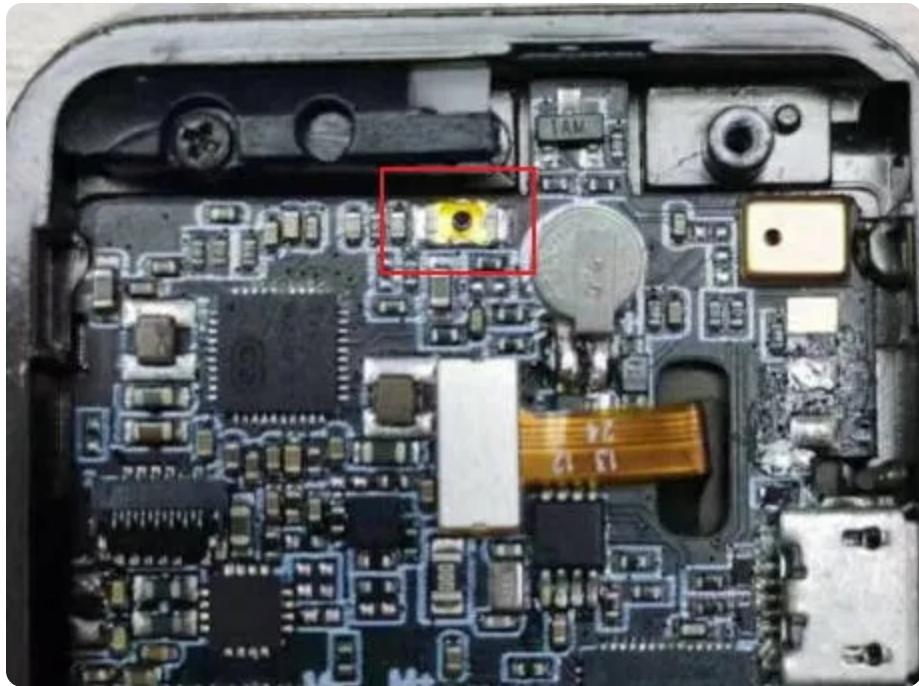
Do not proceed unless an antenna is connected to avoid possible damage to the device's radio.

The following process will manually place the device into the Espressif Firmware Download mode:

Ensure the device is powered off by pressing and holding the crown button.

Remove the device's back cover.

Locate and press and hold the boot button.



While holding the boot button, press the crown button once to turn the device on.

After 2-3 seconds, release the BOOT button.

With the device now in the Espressif Firmware Download mode, you can proceed with flashing using one of the supported flashing methods. It's generally recommended to use the Web Flasher. You can select "T Watch S3" from the device drop-down.

Resources

Firmware file: [firmware-t-watch-X.X.X.xxxxxxx.bin](#)

Purchase Links:

US

Rokland

International
LilyGO
AliExpress



HELTEC® LoRa 32

HELTEC v2.1

WARNING

Not recommended because of design issues! Support is being phased out. Use V3 in new projects.

MCU:

ESP32 (WiFi & Bluetooth)

LoRa Transceiver:

Semtech SX127x

Frequency Options:

433 MHz

868 MHz

915 MHz

Connectors:

Micro USB

Antenna:

U.FL/IPEX antenna connector for LoRa

Features

Built in 0.96 inch OLED display

User and Reset Buttons

No GPS

Resources

Firmware file: [firmware-heltec-v2.1-X.X.X.xxxxxxx.bin](#)

HELTEC v3/v3.1

ⓘ INFO

This device may have issues charging a connected battery if utilizing a USB-C to USB-C cable. It's recommended to use a USB-A to USB-C cable.

MCU:

ESP32-S3FN8 (WiFi & Bluetooth)

LoRa Transceiver:

Semtech SX1262

Frequency Options:

433 MHz

470 - 510 MHz

863 - 870 MHz

902 - 928 MHz

Connectors:

USB-C

Antenna:

Dedicated 2.4 GHz metal spring antenna for WiFi/Bluetooth

U.FL/IPEX antenna connector for LoRa

V3.1 differences

Firmware remains the same as V3 below. Compare schematics:

V3.0 and V3.1. Key differences:

Removal of FDG6322C (a dual N & P channel FET) from the V3.1 power supply.

Antenna filter values in V3.1 (L11 = 1.8pF, C15 = 2.7nH, C24 =

1.8pF) align more closely with ESP32-S3 reference design than V3.0 (L11 = 1.6nH, C15 = 6.9pF, C24 = 2.4pF).

Features

Built in 0.96 inch OLED display

User and Reset Buttons

No GPS

Meshtastic I2C Definitions

SDA: GPIO41

SCL: GPIO42

Pin Map

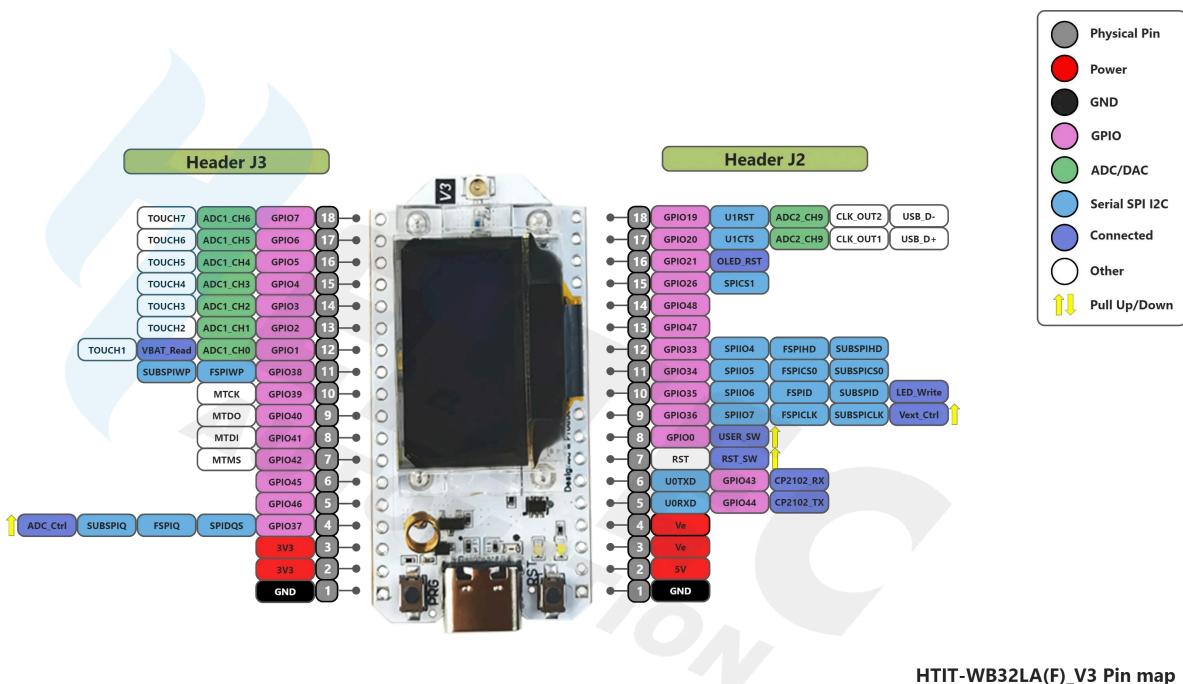


Image Source: Heltec

Resources

Firmware file: `firmware-heltec-v3-X.X.X.xxxxxxx.bin`

Purchase links

Heltec

AliExpress

HELTEC Wireless Stick Lite V3

ⓘ INFO

This device may have issues charging a connected battery if utilizing a USB-C to USB-C cable. It's recommended to use a USB-A to USB-C cable.

MCU:

ESP32-S3FN8 (WiFi & Bluetooth)

LoRa Transceiver:

Semtech SX1262

Frequency Options:

433 - 510 MHz

470 - 510 MHz

863 - 870 MHz

902 - 928 MHz

Connectors:

USB-C

Antenna:

Dedicated 2.4 GHz stamped metal antenna for WiFi/Bluetooth

U.FL/IPEX antenna connector for LoRa (next to the V3 icon)

Features

No display.

User and Reset Buttons

Additional GPIO availability

No GPS

Meshtastic I2C Definitions

SDA: GPIO41

SCL: GPIO42

Pin Map

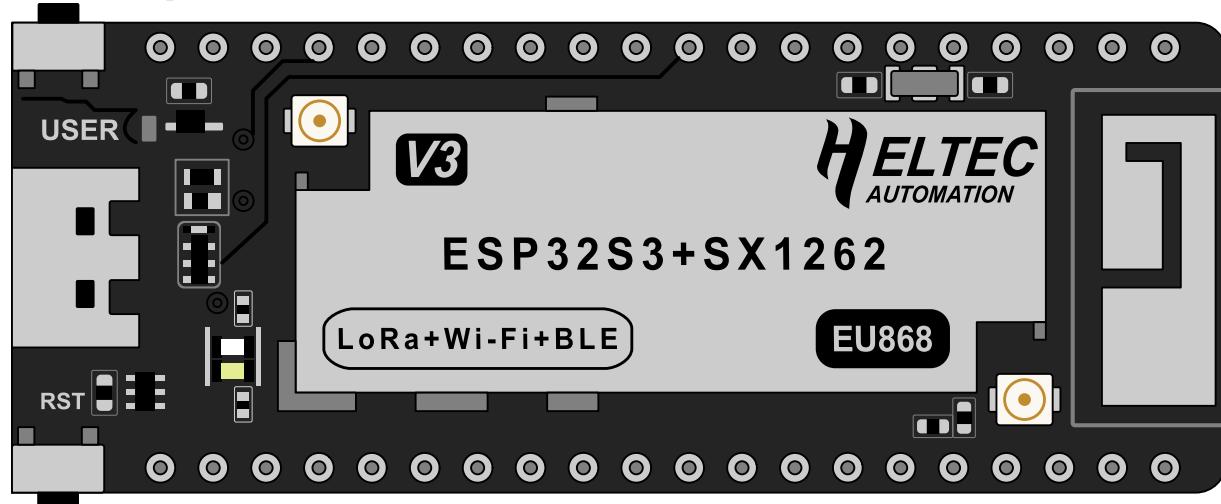


Image Source: Heltec

Resources

Firmware file: [firmware-heltec-wsl-v3-X.X.X.xxxxxxx.bin](#)

Purchase Links:

International

Heltec

AliExpress

HELTEC Tracker

 INFO

This device may have issues charging a connected battery if utilizing a USB-C to USB-C cable. It's recommended to use a USB-A to USB-C cable.

MCU:

ESP32-S3FN8 (WiFi & Bluetooth)

LoRa Transceiver:

Semtech SX1262

Frequency Options:

470 - 510 MHz

863 - 870 MHz

902 - 928 MHz

Connectors:

USB-C

Antenna:

Dedicated 2.4 GHz metal spring antenna for WiFi/Bluetooth

U.FL/IPEX antenna connector for LoRa and GNSS

Features

Onboard 0.96-inch LCD display

User and Reset Buttons

Flashing

If you are having issues flashing the wireless tracker, especially if it's your first attempt, you may need to manually place the device into Espressif's Firmware Download mode. Please follow the process below to do so.

 **WARNING**

Do not proceed unless an antenna is connected to avoid possible damage to the device's radio.

The following process will manually place the device into the Espressif Firmware Download mode:

Unplug the device.

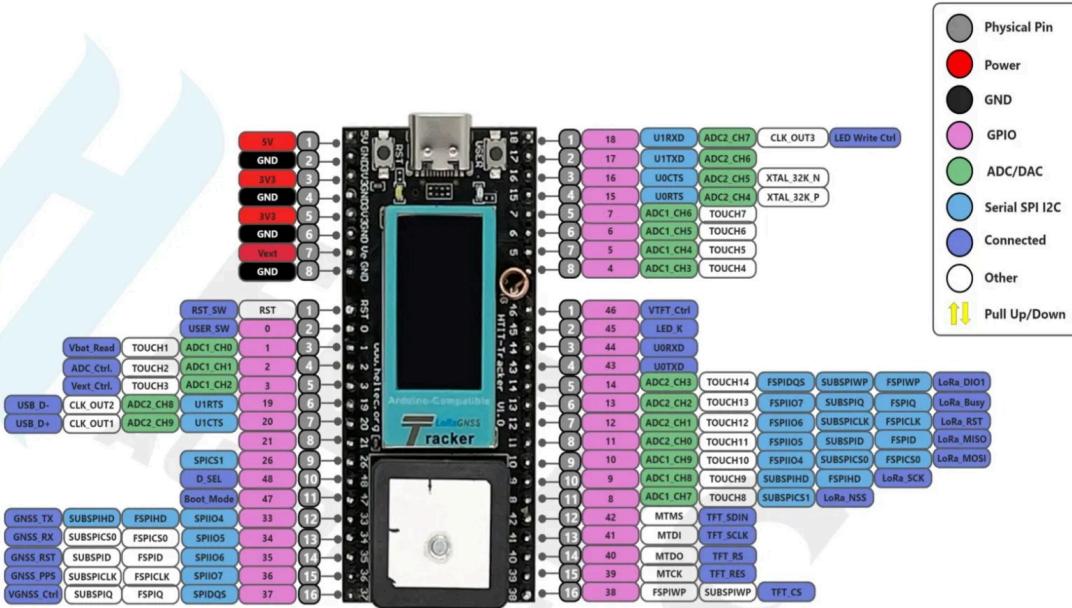
Press and hold the USER button.

Plug device in.

After 2-3 seconds, release the USER button.

With the device now in the Espressif Firmware Download mode, you can proceed with flashing using one of the supported flashing methods. It's generally recommended to use the Web Flasher. You can select "Heltec Wireless Tracker" from the device drop-down.

Pin Map



HT-Tracker_V1 Pin map



Image Source: Heltec

Resources

Firmware file: [firmware-heltec-wireless-tracker-](#)

X.X.X.xxxxxxx.bin

Purchase Links:

International

Heltec

AliExpress

HELTEC Paper



Currently the display is only supported in devices labeled v1.1.

This device may have issues charging a connected battery if utilizing a USB-C to USB-C cable. It's recommended to use a USB-A to USB-C cable.

MCU:

ESP32-S3FN8 (WiFi & Bluetooth)

LoRa Transceiver:

Semtech SX1262

Frequency Options:

470 - 510 MHz

863 - 870 MHz

902 - 928 MHz

Connectors:

USB-C

Antenna:

U.FL/IPEX antenna connector for LoRa

Integrated 2.4 GHz PCB antenna

Features

Onboard 2.13-inch black and white E-Ink display screen

User and Reset switches

No GPS

Resources

Firmware file: [firmware-heltec-wireless-paper-X.X.X.xxxxxxx.bin](#)

Purchase Links:

International

Heltec

AliExpress

Heltec LoRa 32

Hardware Buttons

Functionality

Reset Button (bottom):

Single press: Resets the device.

User/Program Button (top):

Long press: Will signal the device to shutdown after 5 seconds.

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

Triple press: Enables/Disables the GPS Module on demand. If an NPN Transistor is added it will cut power to the GPS board. The NPN pin must be configured on the PIN_GPS_EN inside the Position module in the App for this switching to work.

Heltec ESP32 V3 Supported Peripherals

GPS Module Introduction

This informational guide outlines the process of enhancing the Heltec ESP32 V3 board by integrating the GT-U7 GPS Module. The addition of this module provides precise GPS capabilities and a real-time clock (RTC), eliminating the need for WiFi or a smartphone for time tracking. This enhancement is particularly beneficial for the mesh, where tracking the duration since the last seen device is crucial. However, it's important to note that the GPS module increases the power demand of your node. We'll address this by detailing how to incorporate a switch or an NPN 2N2222 transistor into your setup. This enables firmware-controlled power management, conserving battery life without sacrificing functionality.

It is important to note that the GPS module increases the power demand of the node. This guide details the incorporation of a switch or an NPN 2N2222 transistor to enable firmware-controlled power management, conserving battery life without compromising functionality.

Benefits

GPS Capabilities: Provides the node with the ability to determine

its location with high precision, which is invaluable for tracking, mapping, and various other applications requiring location data.

Real-Time Clock (RTC): Ensures accurate timekeeping on the mesh network without relying on external time sources such as the internet or a connected smartphone.

Power Consumption Considerations

The GT-U7 module is known for its high power consumption, which can potentially shorten the battery lifespan of the node. To mitigate this, two approaches are recommended:

Manual Switch: A simple on/off switch for the GPS module, allowing for manual power management.

NPN 2N2222 Transistor: Facilitates automatic power control through the firmware, enabling the device to turn off the GPS module based on specific conditions or after a set period.

Materials Needed

Heltec ESP32 V3 board

GT-U7 GPS Module

NPN 2N2222 Transistor

Wires and soldering equipment

(Optional) Switch for manual power control

Instructions

Solder a cable from the TXD slot on the GPS module to GPIO 48 on Heltec board. (You may choose your own GPIO pin)

Solder a cable from the RXD slot on the GPS module to GPIO 47 on Heltec board. (You may choose your own GPIO pin)

Solder a cable from the GND slot on the GPS module to GND pin on Heltec board.

Solder a cable from left leg of NPN 2N2222 Transistor to VCC skit on GPS module.

Solder a cable from Right leg of NPN 2N2222 Transistor to 3V/5V pin on Heltec board.

Solder a cable from Middle leg of NPN 2N2222 Transistor to GPIO 48 of Heltec board. (You may choose your own GPIO pin)

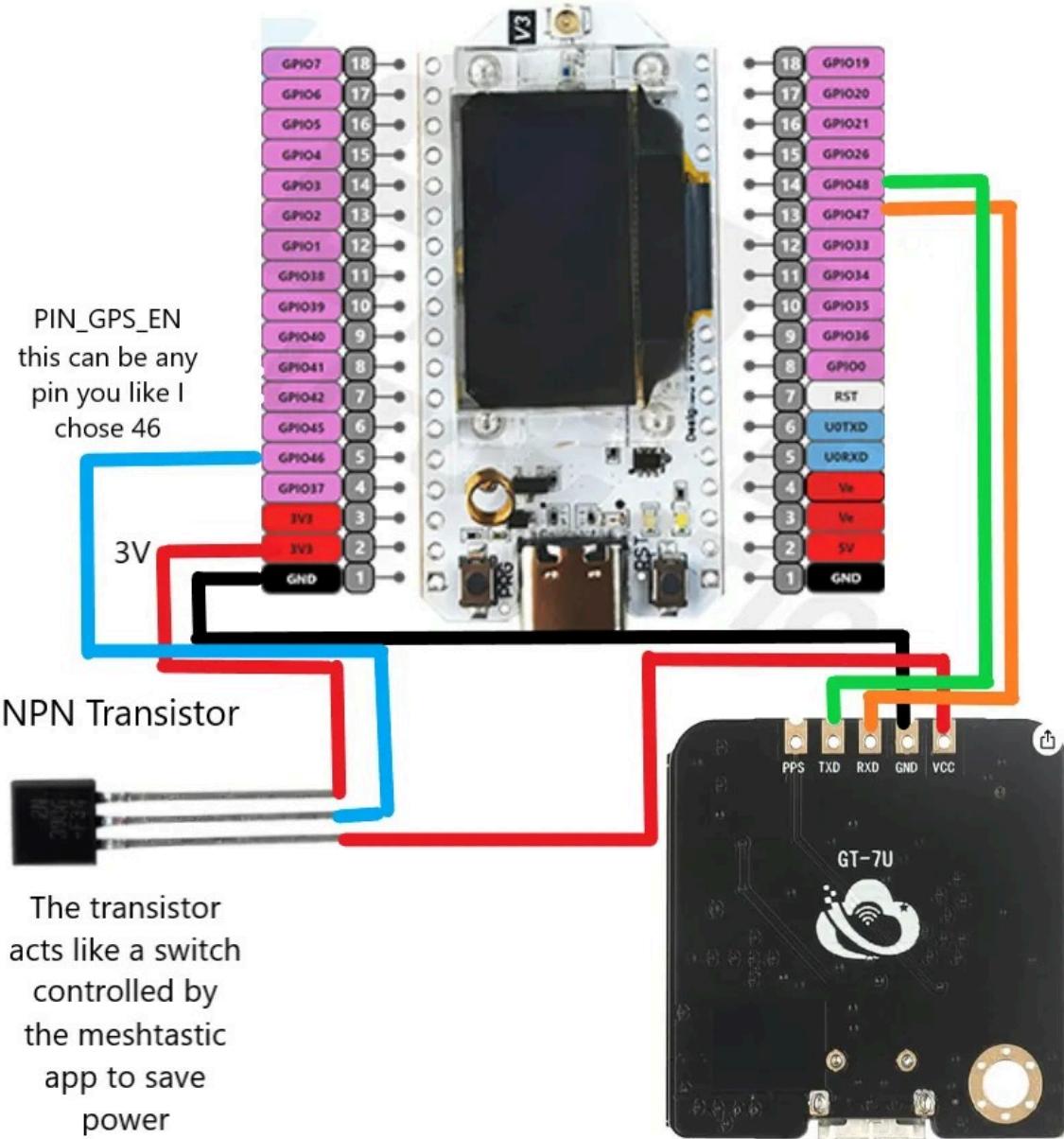
Go to Meshtastic app > Radio Configurations > Position

Set GPS_RX_PIN to 48 (This will communicate to the TXD slot on the GPS)

Set GPS_TX_PIN to 47 (This will communicate to the RXD slot on the GPS)

Set PIN_GPS_EN to 46 (This will allow the meshtastic firmware to turn off the power on the GPS board with the user button of the Heltec Board by pressing it 3 times)

Wiring Diagram



Troubleshooting Tips

If the GPS module does not power on, check the connections to the transistor and ensure that the middle pin is properly configured in the firmware settings. If the message "GPS is Disabled" appears, press the user button on the Heltec board three times to enable it. If the message "No GPS Present" appears, check

that the TXD and RXD are correctly configured. If the message "No GPS Lock" appears, move the node to another location with access to the sky for a bit to lock on properly. If the location is not accurate, walk around to allow the GPS to properly lock on.

Conclusion

By following this guide, you can enhance your Heltec ESP32 V3 board with valuable GPS capabilities and an RTC, while effectively managing power consumption. Whether for a hobby project or a more serious application, these additions significantly expand the potential of your device.

Keyboard

Coming Soon

Buzzer

Coming Soon

Nano Series

The Nano Series, designed by Neil Hao from B&Q Consulting and powered by Meshtastic, are devices engineered to be a portable and durable solution for outdoor adventures such as hiking, piloting, skiing and more. They are designed to strike a balance between RF performance, size, ruggedness, and power efficiency — providing users with a reliable solution in various outdoor environments.

For a full and complete comparison of the different devices in the Nano series, be sure to visit B&Q Consulting's nano series wiki page.

Nano G2 Ultra

The Nano G2 Ultra and Nano G1 Explorer have exactly the same Lora front-end circuit design and internal wideband Lora antenna. The primary design goal of the Nano G2 Ultra is to reduce power consumption and improve battery life by using the low power MCU NRF52840. The typical battery life is optimized to be about 3.5 days. As with the Nano G1 Explorer, the Nano G2 Ultra's design also takes into account the potential impact of the human body on its antenna performance, ensuring optimal RF performance even when carried in a pocket.

Specifications

MCU:

NRF52840

Bluetooth 5.4

LoRa Transceiver:

Semtech SX1262

Frequency Options:

US (902.0Mhz to 928.0Mhz), EU_868 (869.4Mhz to 869.65Mhz), JP (920.8Mhz to 927.8Mhz), ANZ (915.0Mhz to 928.0Mhz), RU (868.7Mhz to 869.2Mhz), KR (920.0Mhz to 923.0Mhz), TW (920.0Mhz to 925.0Mhz), IN (865.0Mhz to 867.0Mhz), NZ_865 (864.0Mhz to 868.0Mhz), TH (920.0Mhz to 925.0Mhz), UA_868 (868.0Mhz to 868.6Mhz), MY_919 (919.0Mhz to 924.0Mhz), SG_923 (917.0Mhz to 925.0Mhz).

Navigation Module:

ATGM336H-5N-71 (Supports GPS, BDS and GLONASS)

Antenna:

New internal wide-band antenna with optimized wide-band LoRa RF frontend circuit.

Connectors:

USB-C

Features

Meshtastic pre-installed.

New internal GPS antenna that significantly reduces GPS lock time compared to the Nano G1. Additionally, the GPS module can be set to a low-power mode with the GPS physical switch.

Integrated enhanced message notification circuit for LED or Buzzer, configurable via physical switches.

Physical power switch to easily turn the device on/off.

Replaceable 1.3" OLED screen with a FPC connector.

Internal Li-Polymer Battery Charger. (*An optional 3.7V 603450 JST 1.25mm Rechargeable Li-Polymer Battery can also be installed.*)

Buck-boost converter to provide stable 3.3V to the system even in the event the Li-Polymer battery voltage drops to as low as 2.5V.

Supply voltage stability is very important to ensure that the performance of RF circuits meets design expectations.

NXP PCF8563 Real Time Clock (RTC)

Optional I2C Extension Board (1x GROVE I2C Socket and 1x SparkFun QWIIC I2C Socket for Sensors and Keyboard)

Resources

Firmware file: [firmware-nano-g2-ultra-X.X.X.xxxxxxx.uf2](#)

Unit Engineering's Official Wiki

Purchase Links:

International

Official Store

Official Tindie Store



Nano G1 Explorer

The Nano G1 Explorer, powered by Meshtastic, is a significant upgrade from the Nano G1. It incorporates the latest RF technologies from B&Q Consulting, featuring a new internal wide-band LoRa antenna that can support frequencies ranging from 815 MHz to 940 MHz. With this new antenna, combined with an optimized wide-band LoRa RF frontend circuit, the Nano G1 Explorer is capable of working with the majority of LoRa frequency bands worldwide without the need for antenna changes. The device's design also takes into account the potential impact of the human body on its antenna performance, ensuring optimal RF performance even when carried in a pocket.

Specifications

MCU:

ESP32 WROOM (WiFi & Bluetooth)

Bluetooth 4.2

LoRa Transceiver:

Semtech SX1262

Frequency Options:

US (902.0Mhz to 928.0Mhz), EU_868 (869.4Mhz to 869.65Mhz), JP (920.8Mhz to 927.8Mhz), ANZ (915.0Mhz to 928.0Mhz), RU (868.7Mhz to 869.2Mhz), KR (920.0Mhz to 923.0Mhz), TW (920.0Mhz to 925.0Mhz), IN (865.0Mhz to 867.0Mhz), NZ_865 (864.0Mhz to 868.0Mhz), TH (920.0Mhz to 925.0Mhz), UA_868 (868.0Mhz to 868.6Mhz), MY_919 (919.0Mhz to 924.0Mhz), SG_923 (917.0Mhz to 925.0Mhz).

Navigation Module:

ATGM336H-5N-71 (Supports GPS, BDS and GLONASS)

Antenna:

New internal wide-band antenna with optimized wide-band LoRa RF frontend circuit.

Connectors:

USB-C

Features

Meshtastic pre-installed.

New internal GPS antenna that significantly reduces GPS lock time compared to the Nano G1. Additionally, the GPS module can be set to a low-power mode with the GPS physical switch.

Integrated enhanced message notification circuit for LED or Buzzer, configurable via physical switches.

Physical power switch to easily turn the device on/off.

Replaceable 1.3" OLED screen with a FPC connector.

Internal Li-Polymer Battery Charger. (*An optional 3.7V 603450 JST 1.25mm Rechargeable Li-Polymer Battery can also be installed.*)

Buck-boost converter to provide stable 3.3V to the system even in the event the Li-Polymer battery voltage drops to as low as 2.5V.

Supply voltage stability is very important to ensure that the performance of RF circuits meets design expectations.

Resources

Firmware file: [firmware-nano-g1-explorer-X.X.X.xxxxxxx.bin](#)

Unit Engineering's Official Wiki

Purchase Links:

International

Official Store

Official Tindie Store



Nano G1

The Nano G1 is the first dedicated hardware device to be designed from scratch purely for Meshtastic by Neil Hao. It has been designed to be small and compact with the inclusion of a high quality internal PCB antenna.

Specifications

MCU

ESP32 WROOM (WiFi & Bluetooth)

Bluetooth 4.2

LoRa Transceiver

Semtech SX1276

Additional ultra-low noise amplifier to improve LoRa receiver sensitivity

Frequency options

US-915 MHz

Navigation Module

ATGM336H-5N-71 (Supports GPS, BDS and GLONASS)

Antenna

Built in 915Mhz Lora PCB Antenna (VSWR <=1.5 @ 915 MHz)

Connectors

USB-C

Features

Meshtastic pre-installed

User button

1.3 inch OLED screen

Buzzer

Resources

Firmware file: `firmware-nano-g1-X.X.X.xxxxxxx.bin`

Unit Engineering's Official Wiki



Nano Series Hardware Buttons

Device

Nano G2 Ultra Buttons

User/Program Button:

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

Long press: Will signal the device to shutdown after 5 seconds (*soft off*).

Message Read Button:

Single press: Clears the New Message LED.

Switches

Power:

Toggle Up: Turns on the device.

Toggle Down: Turns off the device.

Location Pin:

Toggle Up: Sets GPS to operating mode.

Toggle Down: Sets GPS to low power mode.

Bell:

Toggle Up: Selects Buzzer for Enhanced Message Notification Circuit.

Toggle Down: Selects LED for Enhanced Message Notification Circuit.

Nano G1 Explorer

Buttons

User/Program Button:

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

Message Read Button:

Single press: Clears the New Message LED.

Switches

Power:

Toggle Up: Turns on the device.

Toggle Down: Turns off the device.

Location Pin:

Toggle Up: Sets GPS to operating mode.

Toggle Down: Sets GPS to low power mode.

Bell:

Toggle Up: Selects Buzzer for Enhanced Message Notification Circuit.

Toggle Down: Selects LED for Enhanced Message Notification Circuit.

Station G1 device

The Station G1 is the second dedicated hardware device to be designed from scratch purely for Meshtastic Licensed (HAM) Operation by Neil Hao. It has been designed to be small and compact with the inclusion of 35dBm high power PA.

Specifications

MCU

ESP32 WROOM (WiFi & Bluetooth)

Bluetooth 4.2

LoRa Transceiver

Semtech SX1262

Additional 35dBm LoRa Power Amplifier to boost transmit power

Frequency options

US-915 MHz

EU-868 MHz

Navigation Module

ATGM336H-5N-71 (Supports GPS, BDS and GLONASS)

Antenna

SMA Socket

Connectors

USB-C

Features

Meshtastic pre-installed

User button

1.3 inch OLED screen

Optional GPS Module and IO Extension Socket

Optional 12V Battery Docker which can be used as Backup Power,
or in scenarios that require mobility

Resources

Firmware file: [firmware-station-g1-X.X.X.xxxxxxx.bin](#)

Unit Engineering's Official Wiki

Purchase Links:

International

Official Store

Official Tindie Store



Station G1 Hardware Buttons

Functionality

User/Program Button:

Single press: Changes the information page displayed on the device's screen.

Double press: Sends an adhoc ping of the device's position to the network.

Raspberry Pi Pico

The Raspberry Pi Pico series is a range of tiny, fast, and versatile boards built using RP2040, the flagship microcontroller chip designed by Raspberry Pi in the UK.

 **INFO**

Only the Pico W has WiFi/BLE capabilities. Meshtastic supports WiFi on the Pico W (although no web server and HTTP API), but does not currently support BLE.

Specifications

MCU:

Raspberry Pi RP2040

Dual M0+ Core

133MHz CPU Clock

Connectors (On Pico):

Micro-USB on Pico

 **NOTE**

Please be aware that the Raspberry Pi Pico must be used in combination with a Waveshare LoRa Module, which provides the necessary SX1262 LoRa radio. Please ensure to select the

correct frequency for your region.

LoRa Transceiver (On LoRa Module):

SX1262

Frequency Options:

410 - 525 MHz

863 - 870 MHz

902 - 930 MHz

Connectors (On LoRa Module):

U.FL/IPEX antenna connector for LoRa

1.25mm 2-Pin JST for battery

(i) NOTE

LoRa transmissions may interfere with the USB connection. It is recommended to place the antenna as far away from the USB port as possible.

Resources

Firmware file: [firmware-pico-X.X.X.xxxxxxx.uf2](#)

for Pico W use: [firmware-picow-X.X.X.xxxxxxx.uf2](#)

Official Website for the Raspberry Pi Pico, including official reseller links.

Raspberry Pi Pico Supported Peripherals

I²C peripherals

I²C peripherals like OLED Displays (e.g. SSD1306 or SH1106) and keyboards (e.g. CardKB) can be connected to GPIO pins 4 (SDA) and 5 (SCL), which will be recognized on boot. Note that for keyboard input, the Canned Message Module has to be enabled and the input source should be specified.

External device using Serial Module

For connecting an external device via the Serial Module, it's recommended to use GPIO pins 8 (`serial.txd`) and 9 (`serial.rxd`).

LoRa Antennas

Pages

Antenna Testing

Guidance for testing and using your antennas.

Antenna Reports

Community submitted SWR testing.

Antenna Selection

Considerations for choosing an antenna.

Additional Resources

Antenna designs, tools, and coverage simulators.

Community Favorites

These antennas have garnered positive feedback and frequent recommendations within our community. However, we encourage you to conduct your own research to ensure the best choice for your needs. We are not responsible for individual experiences or results.

Compact / Handheld

Frequency	Model	Purchase Links
915 or 868	GIZONT 17cm	AliExpress
915 or 868	GIZONT 20cm	AliExpress
915	LINX ANT-916-CW-HW-SMA	Mouser DigiKey
915	ZIISOR TX915-JKS-20	AliExpress
915	CDEBYTE TX915-JKD-20 (5-pack)	AliExpress

Base Station / Repeater

Frequency	Model	Purchase Links
915	Alfa AOA-915-5ACM	Rokland
915	Rokland 32" 5.8 dBi	Rokland

Automotive

Frequency	Model	Purchase Links
915	Laird MA9-5N	DigiKey
915	Taoglas TI.16.5F11	DigiKey

Antenna Testing

Testing of antennas can be both simple and complex. At its simplest, testing involves sending messages from different locations and seeing which ones are received, and then comparing the results against other antennas. At the complex end, this can be using expensive test chambers and equipment to measure the signal strength, gain, and radiation patterns. However, it seems that a reasonable job can be done with cheaper methods.

If you have sufficient range with your existing aerial, skip this section. If you don't, consider either getting more nodes and / or replace the stock aerial with one tuned (to your region transmitter's frequency):

A quarter wave *tuned* stubby aerial (<10cm for fit-in-pocket) should have a real-world range of a couple of km without significant obstacles (buildings / hills).

Aerial criteria: 50 Ohm, appropriate connector (usually SMA male or U.FL), low VSWR (<2) (at tuning frequency - see its datasheet), gain > 0 dBi .

Caution, avoid suppliers who:

- don't state the aerial's tuned frequency and its specific purpose (LoRa network)
- claim huge gain figures on omni-directional aerials
- don't provide boringly professional datasheets

If you want more range, directionality, or specificity read on.

General guidance

The Meshtastic system is designed to be simple and intuitive to use. However, its LoRa radios rely on point-to-point communications, unit-to-unit, aerial-to-aerial; quite different to the near ubiquitous radio coverage of today's cellphone & Wifi connections.

Some understanding of the factors affecting radio communications will help achieve substantially better service and faster transmission over a greater range with your devices. Here, we'll attempt to provide a top-level set of guidance for use and aerial selection, how to test the aerials, a set of resources for further research, and plenty of opportunity for going deeper.

The Meshtastic devices (of various flavors) lend themselves to experimentation, not only because you can replace their aerials, but also because of their mesh operation. All nodes will, without alteration, relay communications from any other members of the mesh around obstacles and over greater distances. The cost of aerial investment should be weighed against investment in additional low-cost nodes.



While the LoRa devices we are using for Meshtastic are relatively low power radios, care should be taken *not* to operate any radio transmission device without an aerial or with a poorly matched aerial. Radio signals transmitted without an antenna can reflect back and damage the device.

The information collected here is by no means definitive, and necessarily abbreviated (it's a huge topic).

Range Testing

As mentioned, while stating the obvious, the simplest way of performing a test is:

Walk around with a radio sending messages,
For each message, note location and whether 'ACK' ticks are received,
Also, note reported signal strengths,
Change aerials, repeat, and evaluate results.

NOTE

The range test module has been designed for exactly this purpose. It allows one node to transmit a frequent message, and another node to record which messages were received. This data is saved and can be imported to applications such as

Google Earth.

On the topic of testing - performing your own testing and providing feedback is the lifeblood of Meshtastic and open source projects.

Signal Strength Testing

Real world testing is also discussed by Andreas Spiess (the 'guy with the Swiss accent') in his tutorial. He has written code for testing antennas using two Lora32 V1 boards to compare how different antennas behave. Lilygo have also made code available for testing the RSSI on the LORA32 boards and the T-Beam.

Here are a couple of excellent aerial comparisons. Their utility goes beyond the specific aerials tested, giving insight into:

Aerial types & their characteristics,
Testing approaches.

Antenna Matching & Vector Network Analyzers

One of the first things to ensure, is that the antenna you have is tuned to the frequency that you are using. A lot of cheap antennas come labeled with an incorrect working frequency, and this will immediately reduce the emitted signal strength. A Vector Network

Analyzer (VNA) can be used to ensure that the antenna is appropriately matched to the transmission circuit, ensuring that it is operating at the correct impedance, and has a low level of power reflected back from the antenna to the transmitter at the desired transmission frequency.

Andreas Spiess also gives a great explanation of how to use Vector Network Analyzers to correctly tune your antennas, as well as a more in depth tutorial of how to use VNAs. It is important to remember however, that VNAs can only tell you if the antenna is well-matched, not how well it is transmitting. A 50 ohm resistor across the transmitter output would show as ideally matched, but it would be useless at transmitting a signal. There are a number of VNAs now available for less than \$100, making this no longer out of reach for most hobbyists, unlike expensive spectrum analyzers.

Non-aerial Factors Affecting Transmission

Unless you're using your devices in a vacuum, with clear line of sight between aerials the following will have an effect:

- Weather (temperature, humidity, and air pressure),
- Transmission power, bandwidth, spreading factor, and other associated channel factors,
- Number of nodes within reach of the mesh (affects retries)

consequent duty cycle hit),
Absorption by materials (with varying degrees attenuation, by material and depth),
Reflection off surfaces (and channeling through material tunnels, including warm / cold air tunnels commonly present in the atmosphere),
Diffraction around obstacles (over forests and around corners).
Fresnel Zone - A football shape between antennas that must be clear of obstructions or else the signal is attenuated.

Environmental Factors

For a bit of light reading on environmental research:

RF attenuation in vegetation (yes really); if you wander through the woods wondering how your RF is bouncing off leaves dependent on their variety, and wind speed ... well you do, now.
RF attenuation with various building materials.

This one by ITU again is very detailed in its analysis of the drivers of attenuation (I wasn't aware that all EMF radiation exhibits reflection / transmission characteristics akin to light hitting a material boundary. So, depending on the angle of incidence, material and the EMF wavelength, it will be reflected and / or transmitted through).

These RF bands are also made more noisy by adjacent LTE

In summary - wavelengths in Europe fair well in plain sight, curve

over not-so-tall obstacles (including trees), and they reflect off surfaces at low angles of incidence. They go through humans without much attenuation; but not brick, stone, or anything with more attenuation than glass / Kevlar. Oh, and don't sit under an LTE tower and expect it to be plain sailing. RF emissions at adjacent frequencies can interfere at a high enough power.

Discussion

To comment on / join in antenna range Meshtastic discourse

There, you will also find reference to Meshtastic range achievements and aerial recommendations. (Note we've stopped short of making specific supplier aerial recommendations in this wiki.)

LoRa Antenna Testing Reports

RicInNewMexico

RicInNewMexico and others have gone through the trouble of testing a number of commonly purchased antennas in the Meshtastic community and given an opinion on whether or not a given antenna is performing optimally.

Please check out the project on Github: Meshtastic-Antenna-Reports

LoRa Antenna Selection

The stock antennas provided with the T-Beam and other boards usually come from 'mixed bags'. They may not have been designed or tuned for your given frequency range, and they may not be of a quality design.

Matching an antenna to the transceiver frequency is important, as is choosing an appropriate design.

The antenna's design will affect:

- Efficiency - The proportion of the signal which leaves the antenna
- Direction in which the signal is transmitted
- Interference by horizontal or vertical polarization
- Amount of signal which is reflected back to the device itself

CAUTION

While the LoRa devices we use for Meshtastic are relatively low power radios, care should be taken *not* to operate any radio transmission device without an antenna or with a poorly matched antenna. Radio signals transmitted without an antenna can reflect back and damage the device.

Important considerations

What transmission frequency are you using?

Devices on another frequency will not be able to interact with yours. See this listing by The Things Network for frequencies licensed for specific countries.

How will you be carrying / transporting the radio?

A large directional antenna will transmit over significantly greater distance than an omni-directional antenna. However, it must be pointed at its target - so it is not optimal for mobile use.

A tuned half-wave whip antenna may have more omni-directional range than the quarter wave stubby; but it will be conspicuous in your pocket.

Many antennas, especially quarter wave stubby antennas, require the use of ground planes to transmit at peak performance.

Do you want transmission in all directions?

While humans (mostly water) don't attenuate signal greatly (at LoRa frequencies), buildings & walls do. If your antenna is permanently positioned against a building, signal transmitted towards the wall will be largely lost or attenuated.

Does my Meshtastic device have the right power range, impedance, and connector for the antenna?

For the LoRa devices, it should be 50 Ohm impedance with SMA connector. Many antennas will be recommended for LoRa use in their technical details.

In contrast, a close range, contact-less Personal Area Network antenna, or a huge antenna at the end of length of coax designed for a 100W transmitter, are not going to be operable.

Cost, quality, and supply service?

The perfect antenna on paper, sourced from the other side of the world with mixed reviews, doesn't compare to a local supplier who has spent time carefully collating all of the antenna data-sheets for comparison *and* holds stock immediately available. Personally, I prefer to pay significantly more for a time saving, quality service.

How close will the antenna be to my Meshtastic device?

Most cables will significantly degrade the signal strength over any significant distance. It is often more effective to place a node outside than to have it indoors with the antenna outside. The exception might be if there is extreme heat, cold, or humidity, and if the shortest possible low loss cable is used.

Still, a proper enclosure should mitigate bad weather.

Terminology / references

You could do a lot worse than reading the Wikipedia entry for Antenna, along with the Wikipedia entry for LoRa.

Instead of listing the terms, let us recommend this superb tutorial by Andreas Spiess (the 'guy with the Swiss accent').

LoRa Antenna Resources

More antenna information

Hackaday's Introduction to Antenna Basics

An excellent series of presentations on the basics of antenna design and function, presented by spacecraft radio engineer Karen Rucker.

Coverage prediction

Tower Coverage.com

Commercial, but has free options

HeyWhat'sThat

Free with path profiling options

Radio Mobile Online

Radio Mobile Online is a radio wave propagation prediction tool dedicated to amateur radio

RF Tools

Times Microwave Systems

Coaxial Cable Attenuation & Power Handling Calculator

Solwise Link Budget Calculator

Predict the received signal strength

Amateur Radio Toolkit

Android app with lots of antenna information

Antenna designs

1/4 Wave Ground Plane Antenna Calculator

Quadrifilar helicoidal antenna calculator

Building an 868/915 MHz co-linear antenna tutorial

868 MHz antenna schematic PDF

915 MHz antenna schematic PDF

NEC based antenna modeler and optimizer

Solar Powered



Measure Power Consumption

Before you can calculate what size solar panel and battery bank are required you ...

How To Measure Meshtastic Device Power Consumption

Before you can calculate what size solar panel and battery bank are required you need to determine how much power your device consumes. This is an essential and first step in building a solar powered anything.

Setup and Requirements

To measure the average power consumption of a radio device, like a Meshtastic node, requires some hardware and some understanding.

Power Meter

You need a way to measure power. This could be a bench power supply with voltage and amperage readings. It could be a dedicated device like a USB power meter or something else. Most important is that you need to read either volts and amps OR watts over time (over time means it needs to keep track of power used over time, not just instantaneous readings). Also consider if your meter has the resolution to measure as low as you need it to. For

example, some nodes consume 0.005A at 5v. Can your power meter measure and display results that small? The important part here is that your meter needs to keep track of amp hours or watt hours. A normal multimeter won't track this.

What Are We Measuring?

A watt is the unit used to measure power. Use this unit to express how much power your device consumes. If you measure volts and amps you get watts. It's an easy equation. Just multiply volts times amps to get watts: $5v \times 50mA = 250mW$ (also described as $5v \times 0.05A = 0.25W$). 250mW is a measure of consumed power at a specific instance. Technically, it describes the power consumed over one hour. We can't use this number because radio devices like Meshtastic spend a lot of time receiving and only some time transmitting (transmitting consumes a lot more power than receiving). If we measure power when the device is receiving and we say that's how much it consumes in an hour we are not accounting for the less frequent but more power hungry transmissions. We need to get an average over time. To measure power over time we use watt hours. For example, if your device consumes 250mW for one hour and you want to know how much power it consumes in 24 hours, we express it like this: $24h \times 250mW = 6000mWh$. This result is expressed in milliwatt hours (it can also be expressed as: 6Wh).

Duty Cycle

Radios consume relatively little power when receiving. In contrast, they consume a lot more power when transmitting. Duty cycle is the percentage of time the radio transmits. Over a period of time how much of that time is spent transmitting? 5%? 10%? 25%? It's hard to know. It depends on your use-case. Instead of trying to run calculations of receive power vs transmit power based on datasheet numbers, you can run a simulated test and measure power consumption over a period of time. Run your power consumption test for at least 1 hour but the longer you run the test the more accurate it will be (if your test conditions are representative of real life). Consider 2-6 hours as a good test duration.

Test Conditions

Running a test is pretty simple in practice but requires some understanding of how Meshtastic operates in the background.

Background Network Activity

Meshtastic nodes transmit data at regular intervals. These data packets are not messages you might send to other people. They're like beacons; used to tell the mesh your node is operational and on the network. Your node will know if other nodes are online and

where they are (for GPS-enabled nodes). These data packets are sent in the background and are required for the network to function (though some of these can be disabled or configured to transmit less often). Other nodes on the network will respond to some of these beacons with an acknowledgement of receipt. This means there is a baseline of network activity that will consume some power through receiving and transmitting regardless of the messages you manually send.

Simulated Testing

To set up a proper test you need to know or guess how much message traffic your network will have (manual messages sent from a node). Is it 5, 10, 20, or 50 messages in an hour? Only you can know this because everyone's situation is different.

If your message rate is small (like 2-5 messages per hour) it may not impact power consumption very much. The node is already regularly transmitting through background activity so this small number of additional message transmissions will have a low impact on overall power consumption. Your message rate is entirely use-case dependent and no one can tell you what your duty-cycle is. But you can guess and add margin to your calculations to ensure uptime.

Set A Low GPS Location Broadcast Interval

One way to simulate a message in Meshtastic is to increase the

number of times the node sends a GPS location update. Your device does not need a GPS module to use this option. At the time of this writing, the default interval is 2 minutes for a normal node with GPS or 15 minutes for a node with a fixed position. You can decrease this interval and it will broadcast a message more frequently, thus simulating a manual message on the network. You can change this interval on the iPhone app, Android app, web app and CLI. We can use this feature to force more frequent transmissions and thus simulate message traffic. This is helpful because it's automated and doesn't require you to remember to manually send messages during the test.

Fixed Position Broadcast Interval

Most solar base stations don't have a GPS module because it doesn't change location and GPS consumes a lot of power. In Meshtastic you can manually set the GPS coordinates for any node. This is common for solar nodes. When a node has a fixed position it will send out that position update every 15 minutes (by default). That's a baseline of 4 GPS location updates per hour. If you want to simulate 16 messsage transmissions per hour you can set this interval to 3 minutes. That will send an update every 3 minutes which is 20 updates in total. Subtract the baseline 4 updates (that will be present after your test, when deployed) and you get 16 "messages" per hour. For 56 simluated messages per hour you would set this interval to 1 minute.

Example Test Condition With Node Settings

Tests will require at least 2 nodes. The node you measure power from and one other. Though, it may be more realistic to include 3 or 4 nodes on the network. The goal is to be as close to your final deployment network size when measuring power consumption.

These test conditions would simulate a "chatty" network:

3 nodes on the network.

1 "other" node set to broadcast location every 60 seconds (disable smart location).

Test node is paired with phone over Bluetooth.

Test node has a fixed GPS position.

Test node is set to broadcast location every 60 seconds (disable smart location).

Configure all your devices as indicated. It may be helpful to start the test on the hour or on a 15 minute interval (like 2:15). Write the start time on a sticky note and put it next to the test node.

When you stop the test try to do it on the hour or at the 15 minute interval. This makes calculating the results easier. Make sure to reset any previous power measurements on your power meter before starting. Run the test and record the stop time and the total power consumed during the test.

Results

Your results will hopefully be listed in watt hours. Just divide the recorded watt hours by the number of hours for your test to achieve your average power rating in watt hours or miliwatt hours. Save this result and continue to calculating solar panel size.

If your meter measures amp hours you need to convert it.

Convert Amp Hours To Watt Hours

Take your total amp hour result and multiply it by the volts used for the test. For example, your test ran for 3 hours at 5.1v and your meter reads 142mAh. That's $5.1\text{v} * 142\text{mAh} = 724.2\text{mWh}$.

Now divide the total by the test duration: $724.2\text{mWh} / 3\text{h} = 241.4\text{mW}$. This example shows an average power consumption of 241.1mW. Save your test result for the next step, calculating solar panel size.

Software

Android App

3 items

Apple Apps

3 items

Web Client

Meshtastic Web is a Meshtastic client that runs directly in your browser.

Python CLI

2 items

Linux Native

The device software can also run on a native Linux machine thanks to the Portdui...



Integrations

3 items

Android App



Installation

Installation Methods



Usage

Introduction



Translate

How to Contribute

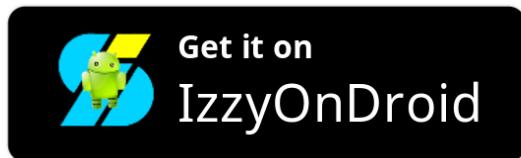
Android Application Installation

Installation Methods

Our Android application is available on our F-Droid repo and Google Play Store. This allows you to connect to your Meshtastic device from your Android phone via Bluetooth, Wi-Fi (if on the same network) or USB On-The-Go (OTG).

The minimum Android version is 5.0 (Lollipop 2014, first BLE support), however Android 6 (Marshmallow 2015) is recommended as Bluetooth is more stable.

Install with F-Droid



Download and Install the F-Droid app from f-droid.org

Open the F-Droid app and navigate to [Settings > Repositories](#).

Click on the icon to add a new repo.

Enter the Meshtastic repo address as follows:

<https://apt.izzysoft.de/fdroid/repo/> as repository address

and

3BF0D6ABFEAE2F401707B6D966BE743BF0EEE49C2561B9BA3907371

1F628937A as fingerprint

Navigate to the [Categories](#) page and refresh (scroll down).

Search for and install the [Meshtastic](#) App.

Install from Play Store



There is a Play Store testing program with the latest cutting edge changes, though this may come with extra bugs.

It is recommended that you follow the Meshtastic Discourse Alpha Testers channel if you decide to join.

Google Play and the Google Play logo are trademarks of Google LLC.

Install by Sideload



The app can also be sideloaded by downloading the .APK from the

Github Releases page.

If you do sideload, you may have to give your browser permissions to run a package installer. If you wish to view the code or contribute to development of the app, please visit the app's GitHub page

Install with Obtainium

Download and Install the Obtainium app from Github.

Open the Obtainium app and navigate to + Add App.

You can easily search for the Android repo with the search field by typing Meshtastic-Android and selecting the Meshtastic/Meshtastic-Android repo. You may also manually enter the Meshtastic Android Github Releases address as follows:

<https://github.com/meshtastic/Meshtastic-Android/releases>.

Under Additional Options for Github toggle as desired Include prereleases* or Fallback to older releases and press Add.

The first time you add an application, obtainium will prompt you for permission to install unknown apps, you will need to toggle Allow from this source and press back. Obtainium will download the Android .APK from the Github release page.

Press Install. Android Installer will prompt "Do you want to install this app?" press Install.

Press Open.

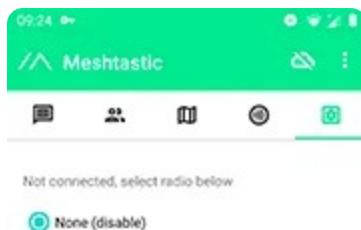
Obtainium allows you to install and update Open-Source Apps directly from their releases pages, and receive notifications when new releases are made available.

*Alpha releases include the latest cutting edge changes which may come with extra bugs. It is recommended that you follow the Meshtastic Discourse Alpha Testers channel if you decide to use these versions.

Android Application Usage

Introduction

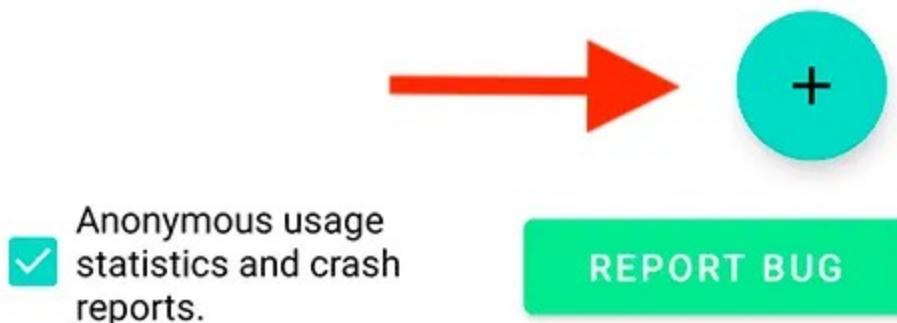
The Meshtastic Android app handles the communication and can show the location of everyone in your private group. Each member of your private mesh can see the location and distance of members and text messages sent to your group chat.



Open the app and you should see the Settings tab like the screen above. Notice the cloud with a slash through it in the upper right, showing no device connection. You can move through the tabs but nothing much will be visible until you connect to a radio device.

Connecting

You will need a device with Meshtastic installed to go any further. See the getting started section for information on how to do this.



To find devices to connect via Bluetooth click the "+" button on the bottom right corner. Devices connected via Wi-Fi or Ethernet using the same network as your phone should be found automatically, or can be manually selected by entering its IP address. If you connect the device via USB OTG to your phone, it will be found automatically.



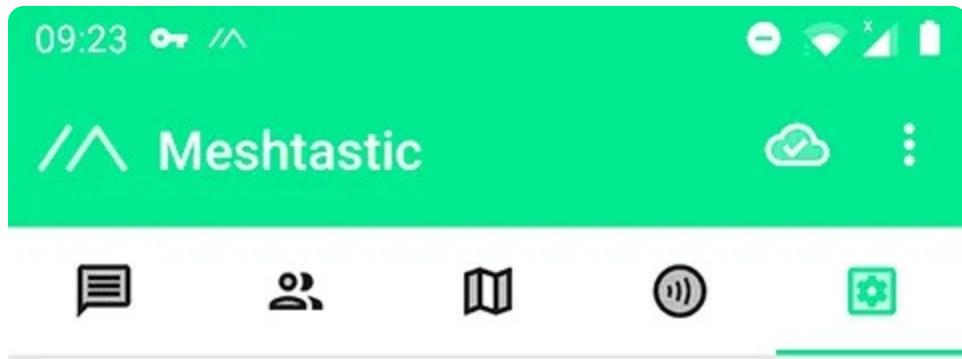
Select the device name, `Meshtastic_769d` in this example. (You will see devices within range, so make sure to get the right one.) Before you can connect for the first time, you need to "pair" the devices to allow communication between them. Some devices are pinless, others require entering a PIN shown on the device screen.

(i) NOTE

If the device was flashed without a screen connected, it will automatically default to a pairing PIN of '123456'. If it was booted with a screen once, the config is set to random pin. If you remove the screen afterwards, it stays like this. Either set it to use the default pin manually, or factory reset it and it will revert to '123456' after the next boot.

This starts the communication with the device. The cloud icon on

the status bar should change and show a check mark.



The cloud icon at the top right corner indicates if you are connected to a device. This currently has three states:



Cloud with a slash through it: No device connected to the application.



Cloud with a tick in it: Device connected to the application.



Cloud with an up arrow in it: Device is connected, but currently sleeping or out of range.

Common tasks

Set your region

In order to start communicating with your mesh, you must select a region. This setting controls which frequency range your device uses and should be set according to your location. See Region Settings for a list of region codes and their meanings.

Tap on the "Region" dropdown in the top-right corner and make the appropriate selection.

Change your name

Edit the "Your name", e.g. to be "Mike Bird". This is the name that other people will see, so make it unique within your group. The initials e.g. "MB" should also be unique and will be used to identify you in the message history and on the device screens. Initials, or "short name", can be customized in the Radio configuration - User settings. The four characters displayed after your initials cannot be changed. These are the last four hex digits of the device MAC address. Devices with unset names will display these four characters as the device short name.



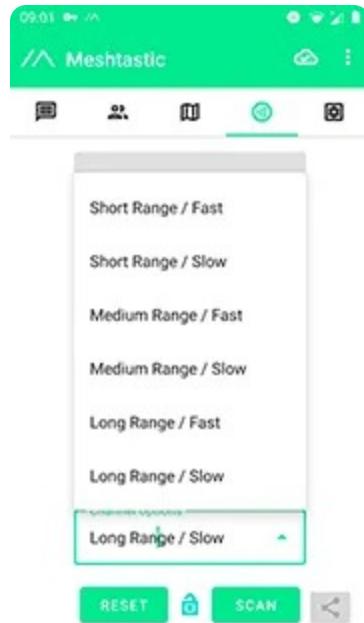
Setup a channel

If you have been sent a QR code or link for Meshtastic, then skip ahead to Join a Channel. Devices have a default channel preconfigured, shown as `#LongFast-I (Long range / Fast)`. It is OK to use this channel if you want your node to be visible to all Meshtastic users (within range) who are using the default channel.

You can also create a new Channel and share the details with your group. The group is private and only those who have the details can join the group and see the messages. You will need to do this once initially, and then only when you want to change or make a new mesh network group.

The Channel tab allows you to create a new private mesh. This screen is initially locked so that you don't change it accidentally.

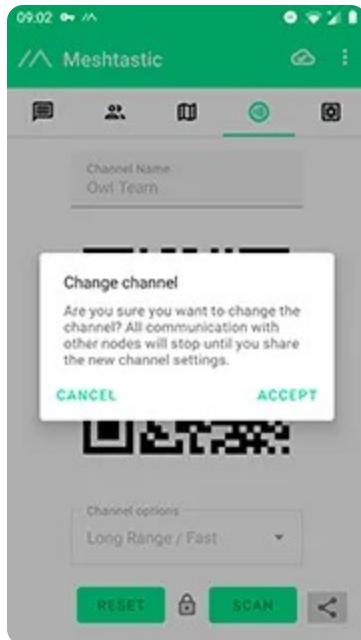
Press the lock symbol, and you will be able to edit. First, select the Channel options, as shown here, and chose the most appropriate option:



Here we selected **Long Range / Fast**, and then made a Channel Name using the keyboard. This identifies your group, here "Owl Team".



You will see a warning because changing the Channel will break communications with your group, i.e. if you change your settings without sharing the new details with the group.



The app will generate a new QR code on the screen. This encodes

the channel details and a random 256-bit key for sharing with the new group. You can share the QR code with other Meshtastic users, or use the Share button and share the link via chat message, SMS, or email. The link is a very long code, for example: <https://www.meshtastic.org/d/#CgUYAyIBAQ>

Join a channel

If another user shares a QR code, you will be able to scan it directly with your camera using the **Scan** button.



Open with Meshtastic

JUST ONCE **ALWAYS**

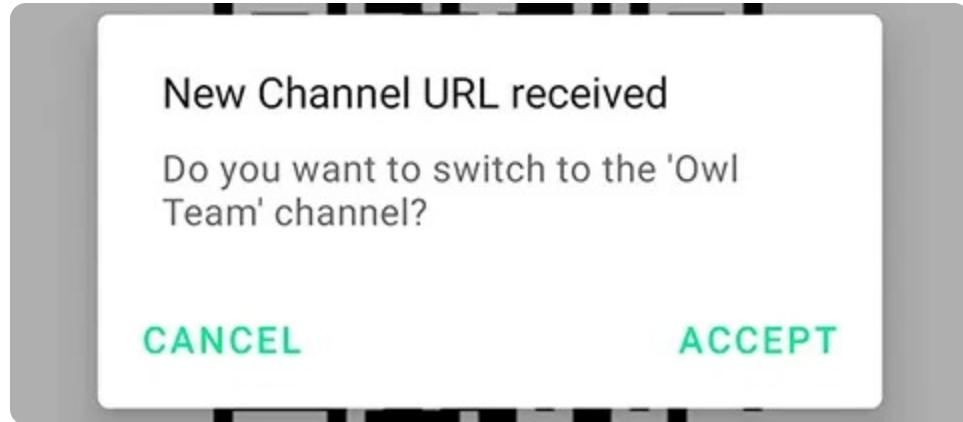
If the channel is shared from a file or link using the **Share** button, you can click on the file or link and you need to choose "Open with Meshtastic".

(!) IF A QR OR URL OPENS A WEBPAGE INSTEAD OF THE APP OR "OPEN WITH MESHTASTIC" IS NOT AN OPTION:

Go to Android Settings > Apps > Default apps > Meshtastic > Opening links

Make sure you have in "links/web address":
www.meshtastic.org

If you see the option "Open the supported links", make sure it is enabled.



Proceed and you should see a message like "Do you want to switch to the 'Owl Team' channel?". Accept this, and the app will change to this new channel. You will lose any current channel setting!

ⓘ NOTE

Setting the same Name and Options directly doesn't work as there are other radio settings (like the unique pre-shared key) encoded in the QR code or link.

You can test changing channels with the QR code shown below.



Send a message



The message window operates like most messaging apps. Note that your primary channel i.e. `LongFast` contact is always shown and works as a group chat. Other contacts are for direct

messaging, or private group chats.

Long press contacts or messages for options, like delete.

Press a node from the Nodes tab to send Direct Messages.

With LoRa (or any radio) there is some uncertainty that the message has been received, so there is a confirmation built-in to the protocol. There are small icons shown to the right of the messages you send:

Cloud with an up arrow: the message is queued in the app, waiting to be handed to the device.

Cloud only: the device received the message from the app, and it has been sent and transmitted via LoRa.

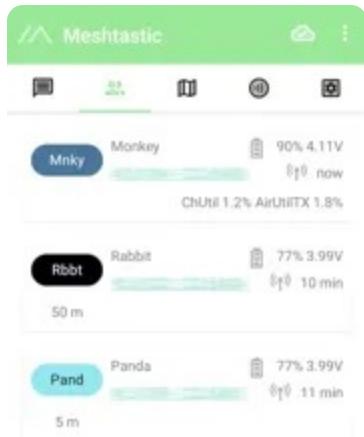
Cloud with a check mark: received at least one node's acknowledgement response. Confirmations could be from any one device.

Person with a check mark - The intended recipient node of your direct message acknowledged the message.

Cloud crossed out: the initial sender did not receive any confirmation within a certain timeout.

By default there is no long-term store-and-forward of messages, so messages not received during transmission are lost.

View your network



The network list shows all the users (devices) that have connected to the same Channel. For each entry, it shows the last time they were active, their location and distance (when available), and their last known power status. In the example above, Monkey is the local user, Rabbit was last heard from 10 minutes ago and is 50m away, and Panda was last heard 11 minutes ago and 5m away.

Tap on a node from the list to start Direct Messaging, request a position update, request a traceroute or add it to your Ignore Incoming Array.

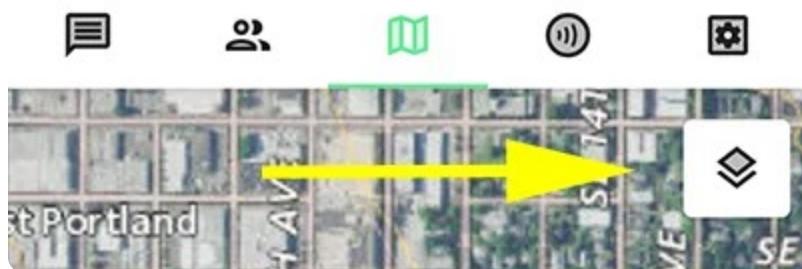
If you have an Admin Channel enabled on your devices, tapping on

the node will also display an option to remotely configure the node.

View the map



The Map tab will show a local map with an icon for each active mesh node that has a known position. The users names are shown above the icon.



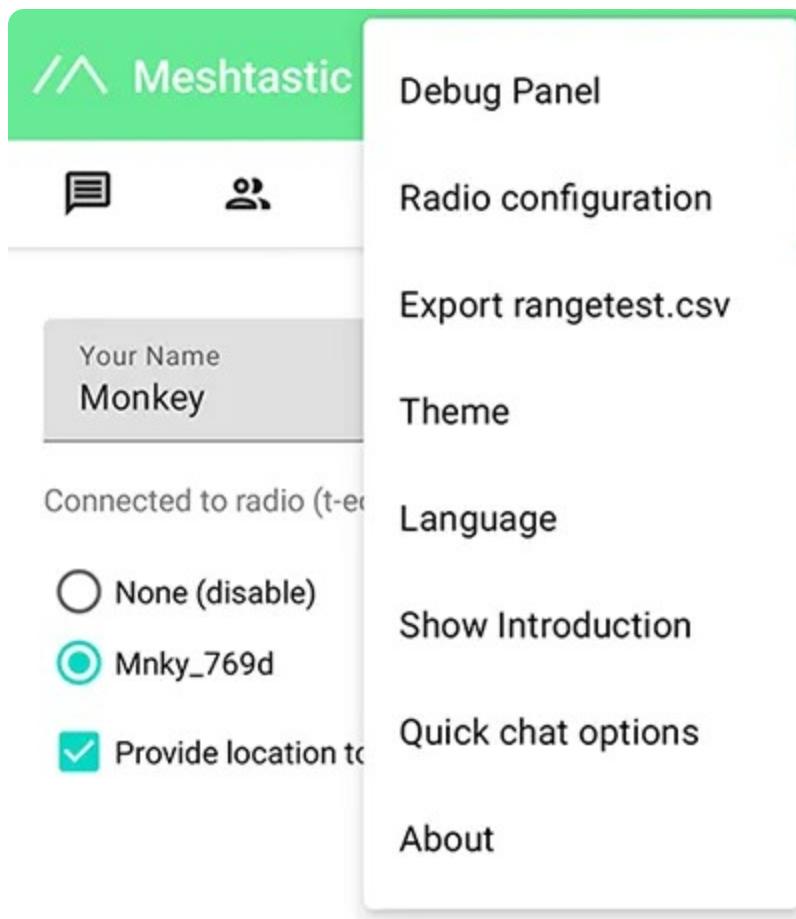
Clicking the layers icon in the top-right will allow you to select the map type.

Offline maps



Some map types allow downloading for offline use. If offline maps are available for your selected map type, a download icon will appear in the bottom-right corner of the map. Tap this icon and choose the option to Download Region, then select the area you wish to download.

Configuration options



Pressing the three vertical dots in the top right corner shows the configuration menu.

Debug Panel



The debug panel allows you to see all packets sent between the application and the device. This can be useful for debugging purposes.

Radio Configuration



Radio Configuration opens a list of all radio and module configuration settings.

See Radio Config for radio settings.

See Module Config for module settings.

At the end of this list are buttons for Reboot, Shutdown, Factory reset, and NodeDB reset.

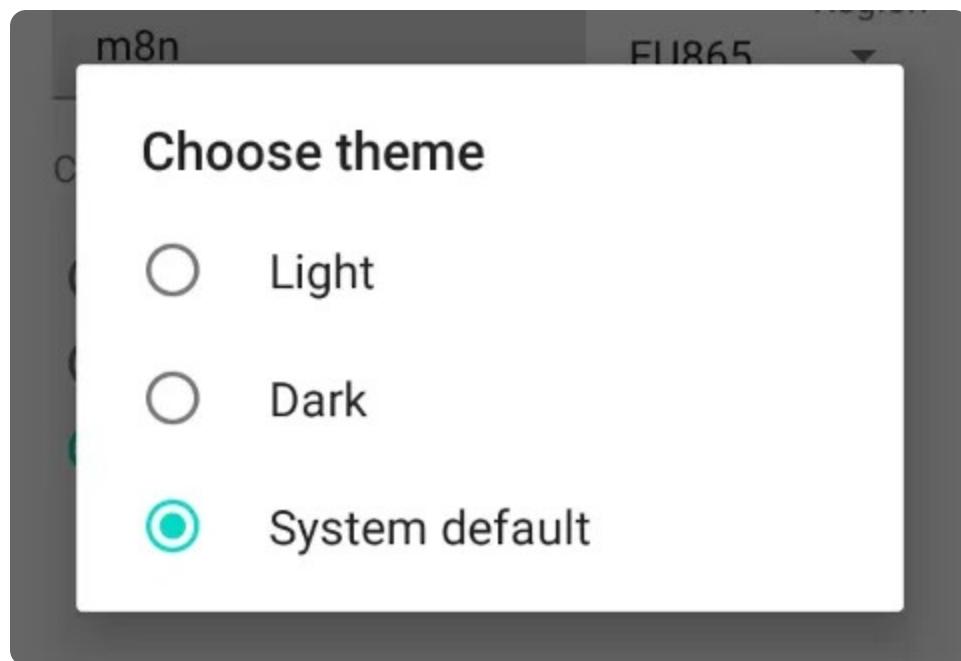
Export rangetest.csv

Allows you to save all your network's position data with GPS coordinates into a .csv (comma separated value) file on your phone. This file can be imported into the spreadsheet application

of your choice for easy viewing. This feature is similar but independent from the device range test module, and results may differ.

To reset the values go to the Debug Panel and press the "Clear" button.

Theme



Allows you to change between light and dark themes, or to select the system default.

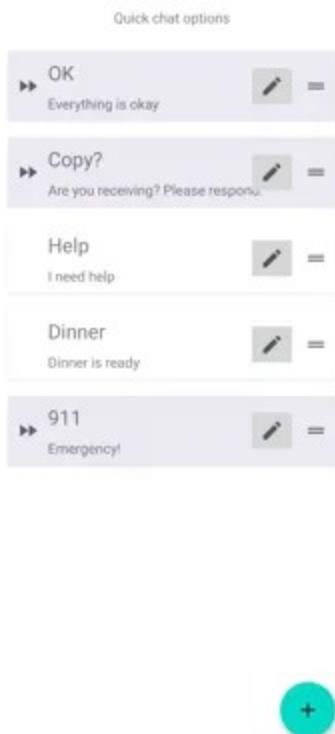
Language

Allows you to select a language for the application's user interface.

Show Introduction

Opens the introduction slideshow.

Quick chat options



Brings up an editor to create and edit quick response messages. These will appear as buttons in the chat window. Messages have the option to send instantly, or be appended to your message and sent manually.

About

Displays the current app version.

Translate the Android App

How to Contribute

Contributing translations to the Meshtastic Android app helps make the project accessible to a wider audience. Follow these steps to add your translations through Crowdin:

Access Crowdin: Visit the Meshtastic project's Crowdin page at <https://crowdin.meshtastic.org>.

Create an Account: Click on 'Sign Up' in the top right corner of the page and follow the prompts to create a Crowdin account.

Navigate to the Project: Once logged in, locate and select the 'Android Application' project from the Crowdin dashboard.

Choose a Language: Find the language you want to contribute translations for and click on 'Go to Editor' to start translating.

Start Translating: In the editor, you'll see a list of strings on the left-hand side. Click on a string to select it, then enter your translation in the editor box. When you're finished with a string, click 'Save' to store your translation. Repeat this process for each string you wish to translate.

Your contribution will be reviewed, and upon approval, your translation will be included in the next release of the Meshtastic Android app. Thank you for helping expand the reach of

Meshtastic!

Apple Applications



Installation

Download on the App Store



Usage

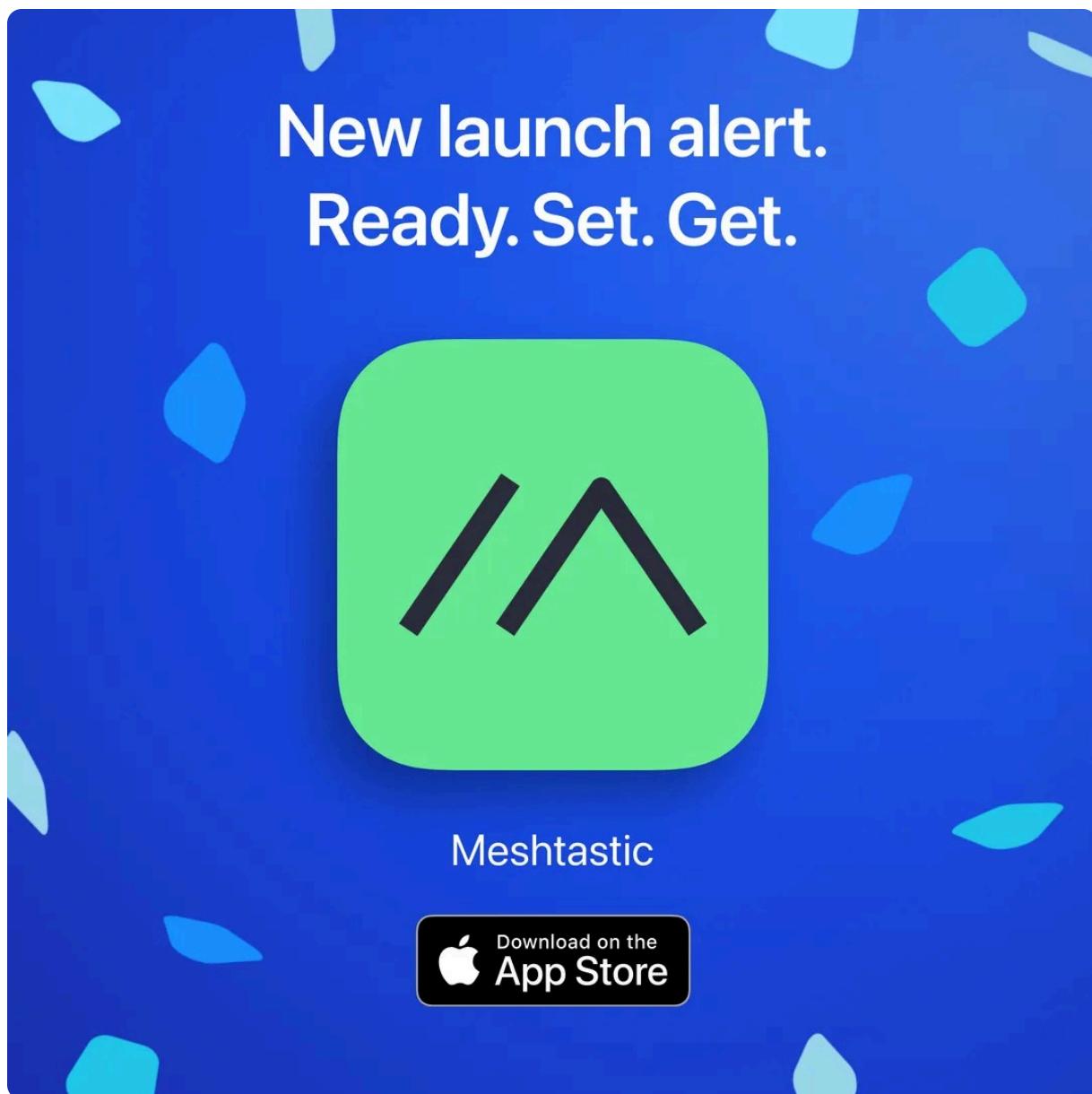
Offline Maps



Translate

How to Contribute

Apple Application Installation



Apple Application Usage

Offline Maps

The Meshtastic app for iOS, iPadOS and macOS supports the sharing of a .mbtiles file with the app for offline map support.

There is an open source cross platform mapping program call QGIS to get a mbtiles out of qgis, start with a small area that you are familiar with. Open qgis, and add a openstreetmap source from the left bar (might be under one of the tile headings). Now you should see something in the main map view. zoom to the area you want.

In the Processing Toolbox (right bar, you may have to show it from the View menu), open Raster Tools > Generate XYZ Tiles (MBTiles)

In Extent, choose Use Map Canvas Extent. This defines the area of the map that will be rendered into the export file Use a zoom level of 12-17. You can play with this later, but thats a good starting point. Note that every increment of maximum zoom will increase the map size 4x. Select JPG if there's aerial/satellite imagery, otherwise just use PNG. That will give the best compression.

Finally choose a location for the output file. Click Run and you

should eventually get a file that should play nice in the Meshtastic app.

Translate the Apple App

How to Contribute

Contributing translations to the Meshtastic Apple app helps make the project accessible to a wider audience. Follow these steps to add translations for a new language:

Fork the Repository: Start by forking the Meshtastic-Apple repository to your GitHub account.

Create a Language Folder: In your forked repository, create a new folder for your language using the language code followed by `.lproj`. For example, for German, create a folder named `de.lproj`.

Copy the Localizable.strings File: Navigate to the English strings folder and copy the `Localizable.strings` file. Paste this file into the folder you created in the previous step.

Translate the Strings: Open the `Localizable.strings` file in your language folder and translate the English strings into your language. Be sure to maintain the format of the file.

Create a Pull Request: Once you've completed the translation, create a new pull request from your forked repository to the main Meshtastic-Apple repository. Title the pull request appropriately (e.g., "Add German Translation") and describe the changes you've

made.

Your contribution will be reviewed, and upon approval, your translation will be included in the next release of the Meshtastic Apple app. Thank you for helping expand the reach of Meshtastic!

Web Client Overview

Meshtastic Web is a Meshtastic client that runs directly in your browser. There are three ways of accessing the app:

Served directly from an ESP32 based node via meshtastic.local or the device's IP Address.¹

A hosted version located at client.meshtastic.org.

Hosting it yourself.

The screenshot shows the Meshtastic Web interface. On the left is a sidebar with navigation icons: Home, Navigation (selected), Map, Config (selected), Channels, Peers, and Config Sections (Device Config selected). Below the sidebar is a node identifier: f3d8 Meshtastic f3d8. The main content area is titled "Device Config" and has tabs for Device, Position, Power, Network, Display, LoRa, and Bluetooth. The "Device" tab is active. Under "Device Settings", there are sections for Role (set to Client), Serial Output Enabled (disabled), and Enabled Debug Log (disabled). Below these are sections for Button Pin (value 0) and Buzzer Pin (value 0). At the bottom is a section for Rebroadcast Mode. A file icon in the top right corner indicates the page can be saved.

Compatibility

The application will work in all major browsers, but specific functionality is limited in some cases. For the best experience we recommend using a Chromium based browser such as Google Chrome or Microsoft Edge.

HTTP

This method of connecting is limited to esp32 devices.

CAUTION

When using the hosted version of the application, all traffic must be served over HTTPS. Meshtastic nodes generate self-signed certificates, and you must inform your browser that you wish to trust the aforementioned certificate. This can be done by first accessing your node directly via your browser:

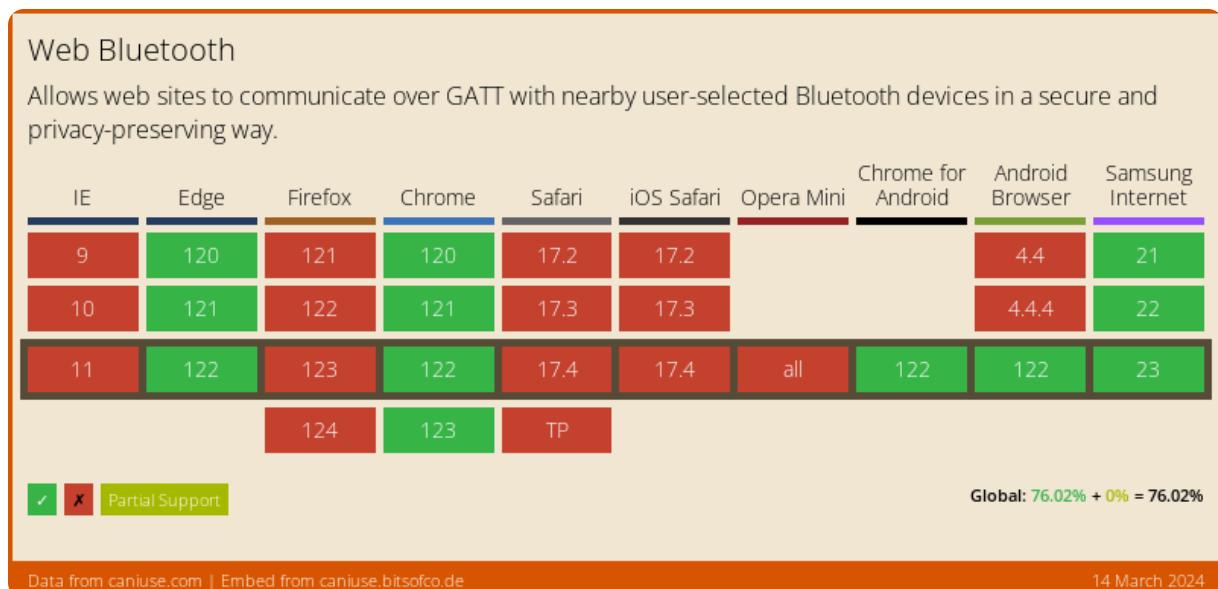
`https://NODE_IP_ADDRESS/` replacing `NODE_IP_ADDRESS`

with the IP address of your node. This can be found on the screen of the device, via your router's DHCP lease page or serial console.

You can access your device over HTTP after you set up your Network Connection

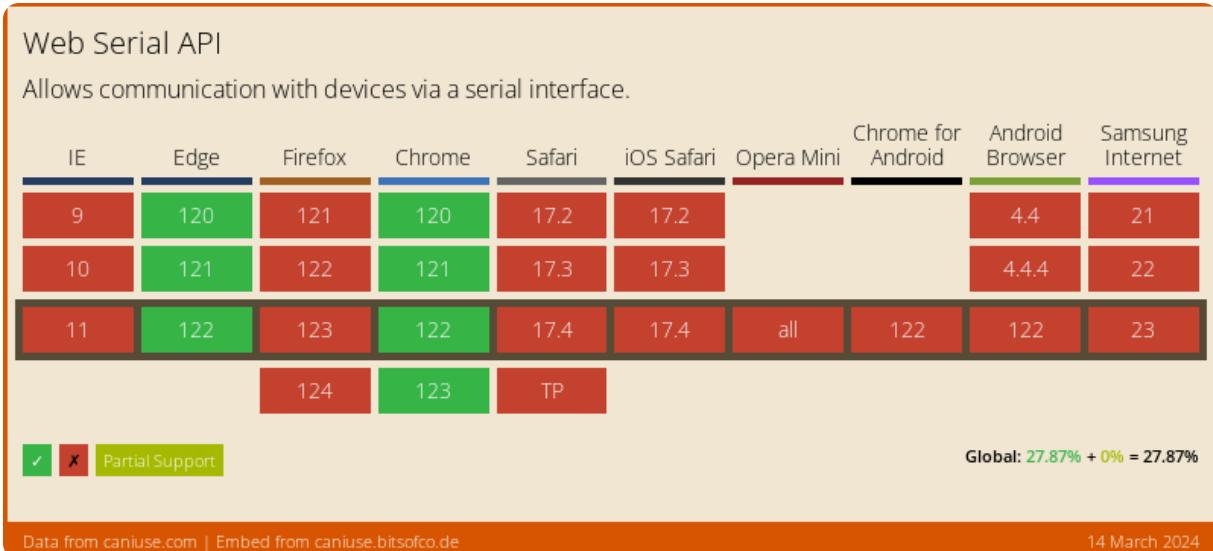
Bluetooth

Bluetooth support is governed by the availability of the Web Bluetooth API as illustrated below. Support is primarily available in Chromium based browsers.



Serial (USB)

The method with the least platform support, which uses the Web Serial API, allows us to connect directly to a Meshtastic node over USB and access it directly within your browser.



Updating

The web interface is included in firmware releases. When a new firmware version is released, the latest WebUI will be automatically bundled.

Self Hosting

The source code for the WebUI can be found on our GitHub

Instructions for building and running the project can be found in the repo's readme.

Footnotes

If multiple Meshtastic devices are serving the web client on the same local area network, these devices can be reached by adding a suffix of "-2" (or higher numbers if there are additional devices)

to the local URL. Example `meshtastic-2.local` or
`meshtastic-3.local`. ↵

Meshtastic Python CLI Guide

The python pip package installs a "meshtastic" command line executable, which displays packets sent over the network as JSON and lets you see serial debugging information from the meshtastic devices. This command is not run inside of python, you run it from your operating system shell prompt directly. If when you type "meshtastic" it doesn't find the command and you are using Windows: Check that the python "scripts" directory is in your path.

Optional Arguments

-h or --help

Shows a help message that describes the arguments.

```
Usage
```

```
meshtastic -h
```

--export-config

Export the configuration of the device. (to be consumed by the '--configure' command).

To create to a file with the connected device's configuration, this command's output must be piped to a yaml file.

Usage

```
meshtastic --export-config > example_config.yaml
```

```
meshtastic --export-config
```

--configure

Configure radio using a yaml file.

Usage

```
meshtastic --configure example_config.yaml
```

--port PORT

The port the Meshtastic device is connected to, i.e. `/dev/ttyUSB0`, `/dev/cu.wchusbserial`, `COM4` etc. if unspecified, meshtastic will try to find it. Important to use when multiple devices are connected to ensure you call the command for the correct device.

Usage

```
meshtastic --port /dev/ttyUSB0 --info  
meshtastic --port COM4 --info
```

--host HOST

The hostname/ipaddr of the device to connect to (over TCP).

Usage

```
meshtastic --host HOST
```

--seriallog SERIALLOG

Logs device serial output to either 'stdout', 'none' or a filename to append to.

Usage

```
meshtastic --port /dev/ttyUSB0 --seriallog
```

--info

Read and display the radio config information.

Usage

```
meshtastic --port /dev/ttyUSB0 --info
```

--set-canned-message

Set the canned message plugin messages separated by pipes | (up to 200 characters).

Usage

```
meshtastic --set-canned-message "I need an  
alpinist!|Call Me|Roger Roger|Keep Calm|On my way"
```

--get-canned-message

Show the canned message plugin message.

Usage

```
meshtastic --get-canned-message
```

--set-ringtone RINGTONE

Set the Notification Ringtone (up to 230 characters).

Usage

```
meshtastic --set-ringtone  
"LeisureSuit:d=16,o=6,b=56:f.5,f#.5,g.5,g#5,32a#5,f5,g#.5,a#.5,32
```

--get-ringtone

Show the stored ringtone.

Usage

```
meshtastic --get-ringtone
```

--nodes

Prints a node list in a pretty, formatted table.

Usage

```
meshtastic --nodes
```

--qr

Displays the QR code that corresponds to the current channel.

Usage

```
meshtastic --qr
```

--get [config_section]

Gets a preferences field.

Configuration values are described in: Configuration.

Usage

```
meshtastic --get lora  
meshtastic --get lora.region
```

--set [config_section].[option] [value]

Sets a preferences field.

Configuration values are described in: Configuration.

Usage

```
meshtastic --set lora.region Unset
```

--seturl SETURL

Set a channel URL.

Usage

```
meshtastic --seturl https://www.meshtastic.org/c/  
GAMiIE67C6zsNm1wQ-KE1tKt0fRKFciHka-  
DShI6G7ElvGOiKgZzaGFyZWQ=
```

--ch-index CH_INDEX

Set the specified channel index.

Usage

```
meshtastic --ch-index 1 --ch-disable
```

--ch-add CH_ADD

Add a secondary channel, you must specify a channel name.

Usage

```
meshtastic --ch-add testing-channel
```

--ch-del

Delete the ch-index channel.

Usage

```
meshtastic --ch-index 1 --ch-del
```

--ch-enable

Enable the specified channel.

Usage

```
meshtastic --ch-index 1 --ch-enable
```

--ch-disable

Disable the specified channel.

Usage

```
meshtastic --ch-index 1 --ch-disable
```

--ch-set CH_SET CH_SET

Set a channel parameter.

Usage

```
meshtastic --ch-set id 1234 --ch-index 0
```

--ch-vlongslow

Change modem preset to `VERY_LONG_SLOW`.

Usage

```
meshtastic --ch-vlongslow
```

--ch-longslow

Change modem preset to `LONG_SLOW`.

Usage

```
meshtastic --ch-longslow
```

--ch-longfast

Change modem preset to (the default) `LONG_FAST`.

Usage

```
meshtastic --ch-longfast
```

--ch-medslow

Change modem preset to `MEDIUM_SLOW`.

Usage

```
meshtastic --ch-medslow
```

--ch-medfast

Change modem preset to `MEDIUM_FAST`.

Usage

```
meshtastic --ch-medfast
```

--ch-shortsow

Change modem preset to `SHORT_SLOW`.

Usage

```
meshtastic --ch-shortsow
```

--ch-shortfast

Change modem preset to `SHORT_FAST`.

Usage

```
meshtastic --ch-shortfast
```

--set-owner `SET_OWNER`

Set device owner name.

Usage

```
meshtastic --set-owner "MeshyJohn"
```

--set-owner-short `SET_OWNER_SHORT`

Set device owner short name (4 characters max).

Usage

```
meshtastic --set-owner-short "MJ"
```

--set-ham SET_HAM

Set licensed Ham ID and turn off encryption.

Usage

```
meshtastic --set-ham KI1345
```

--dest DEST

The destination node id for any sent commands. Used for Remote Node Administration

Usage

```
meshtastic --dest '!28979058' --set-owner "MeshyJohn"
```

--sendtext SENDTEXT

Send a text message. Can specify a channel index ('--ch-index') or a destination ('--dest').

Usage

```
meshtastic --sendtext "Hello Mesh!"
```

--sendping

Send a ping message (which requests a reply).

Usage

```
meshtastic --sendping
```

--traceroute TRACEROUTE

Traceroute from connected node to a destination. You need pass the destination ID as an argument. Only nodes that have the encryption key can be traced.

Usage

```
meshtastic --traceroute '!ba4bf9d0'
```

--request-telemetry

Request telemetry from a node. You need to pass the destination ID as an argument with '--dest'. For repeaters, the nodeNum is required.

Usage

```
meshtastic --request-telemetry --dest '!ba4bf9d0'  
meshtastic --request-telemetry --dest 1828779180
```

--ack

Used in combination with --sendtext to wait for an acknowledgment.

Usage

```
meshtastic --sendtext "Hello Mesh!" --ack
```

--reboot

Tell the node to reboot.

Usage

```
meshtastic --reboot
```

--shutdown

Tell the node to shutdown.

Usage

```
meshtastic --shutdown
```

--factory-reset

Tell the node to install the default config.

Usage

```
meshtastic --factory-reset
```

--reset-nodedb

Tell the node to clear its list of nodes.

Usage

```
meshtastic --reset-nodedb
```

--reply

Reply to received messages.

Usage

```
meshtastic --reply
```

--gpio-wrb GPIO_WRB GPIO_WRB

Set a particular GPIO # to 1 or 0.

Usage

```
meshtastic --port /dev/ttyUSB0 --gpio-wrb 4 1 --dest  
'!28979058'
```

--gpio-rd GPIO_RD

Read from a GPIO mask.

Usage

```
meshtastic --port /dev/ttyUSB0 --gpio-rd 0x10 --dest  
'!28979058'
```

--gpio-watch GPIO_WATCH

Start watching a GPIO mask for changes.

Usage

```
meshtastic --port /dev/ttyUSB0 --gpio-watch 0x10 --  
dest '!28979058'
```

--no-time

Suppress sending the current time to the mesh.

Usage

```
meshtastic --port /dev/ttyUSB0 --no-time
```

--setalt SETALT

Set device altitude (allows use without GPS).

Usage

```
meshtastic --setalt 120
```

--setlat SETLAT

Set device latitude (allows use without GPS).

Usage

```
meshtastic --setlat 25.2
```

--setlon SETLON

Set device longitude (allows use without GPS).

Usage

```
meshtastic --setlon -16.8
```

--debug

Show API library debug log messages.

Usage

```
meshtastic --debug --info
```

--test

Run stress test against all connected Meshtastic devices.

Usage

```
meshtastic --test
```

--ble BLE

Connect to a Meshtastic device using its BLE address or name. This option allows for wireless communication with the device, similar to how the `--host` option is used for TCP connections.

Usage

```
meshtastic --ble "device_name_or_address" --info
```

--ble-scan

Scan for available Meshtastic devices using BLE. This command lists discoverable devices, providing a convenient method to identify devices for connection via BLE.

Usage

```
meshtastic --ble-scan
```

--noprotocol

Don't start the API, just function as a dumb serial terminal. Probably not very helpful from the command line. Used more for testing/internal needs.

Usage

```
meshtastic --noproto
```

--version

Show program's version number and exit.

```
Usage
```

```
meshtastic --version
```

--support

Print out info that would be helpful supporting any issues.

```
Usage
```

```
meshtastic --support
```

Meshtastic Python CLI installation

This library provides a command line interface (CLI) for managing the user settings of Meshtastic nodes and provides an easy API for sending and receiving messages over mesh radios. Events are delivered using a publish-subscribe model, and you can subscribe to only the message types you are interested in.

The Meshtastic-python repo and API documentation are excellent sources of information.

If you wish to view the code or contribute to development of the python library or the command line interface, please visit the Meshtastic python GitHub page.

There are standalone executables for Windows and Ubuntu if you do not want to install python and/or the python libraries required to run the meshtastic CLI tool. See Standalone for more information.

Installation can also be easily done through the Python package installer pip:

 **NOTE**

You must use pip version 20 or later. To upgrade to the latest pip, do: `pip install --upgrade pip`

(!) INFO

Make sure that the `PATH` variable also gets installed by checking the box while installing python. If you don't, python may not be available and you may not be able to call `meshtastic` from your CLI. If you do forget to check that box, you will need to install the path environment variable for python on your operating system.

(!) IMPORTANT

You may need to install a driver from Silicon Labs for the CP210X USB to UART bridge
Some newer boards may require the drivers for the CH9102 or Direct Download for Windows 7.

Linux

Check that your computer has the required serial drivers installed
Connect your Meshtastic device to your USB port
Use the command

```
lsusb
```

You should see something like:

```
ID 10c4:ea60 Silicon Labs CP210x UART Bridge for CP210X  
ID 1a86:55d4 QinHeng Electronics USB Single Serial for  
CH9102
```

Check that your computer has Python 3 installed.

Use the command

```
python3 -V
```

If this does not return a version, install python

```
sudo apt-get update  
sudo apt-get install python3
```

Pip is typically installed if you are using python 3 version ≥ 3.4

Check that pip is installed using this command

```
pip3 -V
```

If this does not return a version, install pip

```
sudo apt-get install python3-pip
```

Optional: use a python virtual environment (otherwise jump to step "Install pytap2")

Install python-virtualenvwrapper (arch based distros as an example)

```
sudo pacman -Syu python-virtualenvwrapper
```

Create a virtual environment

```
source /usr/bin/virtualenvwrapper.sh  
mkvirtualenv meshtastic  
workon meshtastic
```

Install pytap2

```
pip3 install --upgrade pytap2
```

Install meshtastic:

```
pip3 install --upgrade meshtastic
```

macOS

Check that your computer has the required serial drivers installed

Connect your Meshtastic device to your USB port

Navigate to Apple Menu ◊ > About This Mac > System

Report... > Hardware > USB

You should see something like CP210X USB to UART Bridge

Controller

If not download the drivers from Silicon Labs.

Check that your computer has Python 3 installed.

Use the command

```
python3 -V
```

If this does not return a version, install python

The following uses Homebrew to install `python3` which includes `pip3`.

Check if you have Homebrew installed with the following command

```
brew -V
```

If it's not installed, follow the instructions on the Homebrew website before continuing.

Install Python3

```
brew install python3
```

Pip is typically installed if you are using python 3 version ≥ 3.4

Check that pip is installed using this command

```
pip3 -V
```

If this does not return a version, install pip

Install pytap2

```
sudo pip3 install --upgrade pytap2
```

Install meshtastic:

```
sudo pip3 install --upgrade meshtastic
```

Windows

Check that your computer has the required serial drivers installed

Connect your Meshtastic device to your USB port

Open Device Manager

Under **Ports (COM & LPT)** you should see something like

Silicon Labs CP210X USB to UART Bridge (COM5)

If not download the drivers from Silicon Labs or use the direct link below.

WARNING

You must install the CP210x Universal Windows Driver. If you do not install this driver, your device may not work and the driver may need to be uninstalled from device manager before installing the correct driver.

Check that your computer has Python 3 installed.

Use the command

```
py -V
```

If this does not return a version, install python

Pip is typically installed if you are using python 3 version ≥ 3.4

Check that pip is installed using this command

```
pip3 -V
```

If this does not return a version, install pip

Install pytap2

```
pip3 install --upgrade pytap2
```

Install meshtastic:

```
pip3 install --upgrade meshtastic
```

Termux

NOTE

Wifi connection is currently under development and may not be working properly just yet. If you would like to provide feedback or test this feature, please visit our forum or join our Discord server for more information.

Install Termux from the F-Droid app store (Google play does not currently support the latest builds)

Load Termux and update the package list

```
pkg update
```

Upgrade the installed packages

```
pkg upgrade
```

Install python

```
pkg install python
```

Upgrade pip and installed meshtastic and some of its dependencies

```
pip install --upgrade pip pygatt pytap2 wheel  
meshtastic
```

 **NOTE**

Be aware that the Meshtastic CLI is not able to control the nodes over USB through termux, but you can control devices over Wifi using the `--host x.x.x.x` option with the device IP address. However, only ESP32 devices can use Wifi currently.

 **INFO**

You may need to close and re-open the CLI. The path variables may or may not update for the current session when installing.

Standalone

There are standalone executable files for Windows and Ubuntu. A single file is all you need to run the command line interface (CLI) Meshtastic tool. There is a zip file per operating system. To use, see the operating system specific notes below:

They can be found on the Releases page.

Ubuntu

Download meshtastic_ubuntu

Run the following command to make the file executable and rename it 'meshtastic':

```
chmod +x meshtastic_ubuntu && mv meshtastic_ubuntu  
meshtastic
```

To run the cli:

```
./meshtastic
```



TIP
Copy (or move) this binary somewhere in your path.

Windows

Download meshtastic_windows

Rename to meshtastic.exe

To run, open a windows command prompt, navigate to the location of the executable and run:

```
meshtastic.exe
```

Using the Meshtastic CLI

This section covers using the "meshtastic" command line executable, which displays packets sent over the network as JSON and lets you see serial debugging information from the Meshtastic devices.

NOTE

The `meshtastic` command is not run within python but is a script run from your operating system shell prompt. When you type "meshtastic" and the prompt is unable to find the command in Windows, check that the python "scripts" directory is in your path.

Viewing Serial Output

The `--noprotocol` command in the Meshtastic Python CLI is used to disable the API and function merely as a "dumb serial terminal." This mode of operation allows both the API and device functionalities to remain accessible for regular use, while simultaneously providing a window into the raw serial output. This feature can be particularly useful for debugging, development, or understanding the low-level communication between devices.

Example Usage

```
user@host % meshtastic --noprot
# You should see results similar to this:
WARNING file:mesh_interface.py _sendToRadio line:681
Not sending packet because protocol use is disabled
by noProto
Connected to radio
WARNING file:mesh_interface.py _sendPacket line:531
Not sending packet because protocol use is disabled
by noProto
INFO | 18:38:04 711 [DeviceTelemetryModule]
(Sending): air_util_tx=0.116361,
channel_utilization=1.916667, battery_level=101,
voltage=4.171000
DEBUG | 18:38:04 711 [DeviceTelemetryModule]
updateTelemetry LOCAL
DEBUG | 18:38:04 711 [DeviceTelemetryModule] Node
status update: 2 online, 4 total
INFO | 18:38:04 711 [DeviceTelemetryModule] Sending
packet to phone
INFO | 18:38:04 711 Telling client we have new
packets 28
```

Getting a list of User Preferences

You can get a list of user preferences by running '--get' with an invalid attribute such as 'all'.

```
meshtastic --get all
```

Changing settings

You can also use this tool to set any of the device parameters which are stored in persistent storage. For instance, here's how to set the device to keep the Bluetooth link alive for eight hours (any usage of the Bluetooth protocol from your phone will reset this timer)

Expected Output

```
# You should see a result similar to this:  
mydir$ meshtastic --set power.wait_bluetooth_secs  
28800  
Connected to radio...  
Setting power.wait_bluetooth_secs to 28800  
Writing modified preferences to device...
```

Or to set a node at a fixed position and never power up the GPS.

```
meshtastic --setlat 25.2 --setlon -16.8 --setalt 120
```

Or to configure an ESP32 based board to join a Wifi network as a station:

```
meshtastic --set network.wifi_ssid mywifissid --set  
network.wifi_psk mywifipsw --set network.wifi_enabled  
1
```

 **NOTE**

For a full list of preferences which can be set (and their documentation) can be found in the protobufs.

Changing channel settings

The channel settings can also be changed, either by using a standard (shareable) meshtastic URL or you can set a particular channel parameter (for advanced users).

 **WARNING**

Meshtastic encodes the radio channel and PSK in the channel's URL. All nodes must connect to the channel again by using the URL provided after a change in this section by performing the **--info** switch.

```
meshtastic --ch-set name mychan --ch-index 1 --info
```

You can even set the channel preshared key to a particular AES128 or AES256 sequence.

Use `--ch-set psk none --ch-index 0` to turn off encryption.

Use `--ch-set psk random --ch-index 0` to assign a new (high quality) random AES256 key to the primary channel (similar to what the Android app does when making new channels).

Use `--ch-set psk default --ch-index 0` to restore the standard 'default' (minimally secure, because it is in the source code for anyone to read) AES128 key.

All `ch-set` commands need to have the `ch-index` parameter specified:

```
meshtastic --ch-index 1 --ch-set name mychan --info
```

Ham radio support

Meshtastic is designed to be used without a radio operator license. If you do have a license you can set your operator ID and turn off encryption with:

Expected Output

```
# You should see a result similar to this:  
mydir$ meshtastic --set-ham KI1345  
Connected to radio  
Setting Ham ID to KI1345 and turning off encryption  
Writing modified channels to device
```

Toggling `set-ham` changes your device settings in the following ways.

Setting	<code>set-ham</code> Default	Normal Default
<code>IsLicensed</code>	<code>true</code>	See User Config - <code>IsLicensed</code>
<code>LongName</code>	<i>Your CallSign</i>	See User Config - <code>LongName</code>
<code>ShortName</code>	<i>Abrv CallSign</i>	See User Config - <code>ShortName</code>
<code>PSK</code>	<code>""</code>	See Channel Settings - <code>PSK</code>

Changing the preshared key

You can set the channel preshared key to a particular AES128 or AES256 sequence.

```
meshtastic --ch-set psk
```

Use "--ch-set psk none" to turn off encryption.

Use "--ch-set psk random" will assign a new (high quality) random AES256 key to the primary channel (similar to what the Android app does when making new channels).

Use "--ch-set psk default" to restore the standard 'default' (minimally secure, because it is in the source code for anyone to read) AES128 key.

All "ch-set" commands will default to the primary channel at index 0, but can be applied to other channels with the "ch-index" parameter.

Utilizing BLE via the Python CLI

The Python CLI supports communicating with Meshtastic devices via Bluetooth Low Energy (BLE), in addition to the standard serial and TCP/IP connections. To use BLE, you will need a Bluetooth adapter on your computer.

Scan for BLE Devices

First, you can scan for available Meshtastic devices using:

```
meshtastic --ble-scan
```

This will list all Meshtastic devices discoverable over BLE along with their addresses and names in the following format:

```
Found: name='Meshtastic_1234'  
address='AA11BB22-CC33-DD44-EE55-FF6677889900'  
BLE scan finished
```

Available Commands

Once you have the device address or name, you can utilize it alongside your normal Python CLI commands like `--info`, `--nodes`, `--export-config`, etc. but with the `--ble` option to communicate via BLE rather than serial.

You can use **either** the name or address to issue your commands.

```
meshtastic --ble <name> --info  
meshtastic --ble <address> --nodes
```

The initial time you use the `--ble` option for a specific device, you will be prompted to enter the BLE PIN code (as is normal with a client). Once paired, this step won't be required unless you forget the device.

NOTE

On Linux, you may need to pair the BLE device using

`bluetoothctl` before connecting. This allows entering the required PIN for pairing.

Additional BLE Examples

Scan for devices and get info from the first one:

```
meshtastic --ble-scan
# Sample output:
# Found: name='Meshtastic_1234'
address='AA11BB22-CC33-DD44-EE55-FF6677889900'
# Found: name='Meshtastic_5678'
address='FF00DD00-AA11-BB22-CC33-DD44EE5566FF'
BLE scan finished

meshtastic --ble AA11BB22-CC33-DD44-EE55-FF6677889900
--info
```

Connect to a named device and read the node list:

```
meshtastic --ble Meshtastic_1234 --nodes
```

Export device config with --export-config

```
meshtastic --ble Meshtastic_1234 --export-config >
config.yaml
```

Send a command to a remote device using the --dest option:

```
meshtastic --dest '!fe1932db4' --set  
device.is_managed false --ble Meshtastic_9abc
```

**For debugging, you can enable verbose BLE logging by adding the
--debug flag:**

```
meshtastic --ble AA11BB22-CC33-DD44-EE55-FF6677889900  
--debug --info
```

FAQ/common problems

This is a collection of common questions and answers from our friendly forum.

Permission denied: '/dev/ttyUSB0'

As previously discussed on the forum

This indicates an OS permission problem for access by your user to the USB serial port. Typically this is fixed by the following.

```
sudo usermod -a -G dialout <username>
```

If adding your user to the dialout group does not work, you can use the following command to find out which group to add your

user to. In this example (from Arch Linux) the group was "uucp"

```
> ls -al /dev/ttyACM0
crw-rw---- 1 root uucp 166, 0 Jul 20 21:52 /dev/
ttyACM0
```

Mac OS Big Sur

There is a problem with Big Sur and pyserial. The workaround is to install a newer version of pyserial:

```
pip3 install -U --pre pyserial
```

Linux Native Application

The device software can also run on a native Linux machine thanks to the Portduino framework.

The application either simulates some of the interfaces, or uses the real hardware of your machine. Device firmware from 1.3.42 and on even allows you to simulate the LoRa chip by sending and receiving Meshtastic packets via a local TCP port. In this way, you can let multiple instances of the application communicate with each other as if they did via LoRa. For instructions on how to use it, see the interactive simulator that also emulates a wireless environment using simulated positions of the nodes.

Usage with a Linux machine

The easiest way of building the native application is using Visual Studio Code with the PlatformIO extension. See the instructions for creating such a building environment [here](#).

Then after opening the firmware repository in Visual Studio Code, simply click on the PlatformIO extension in the left bar, select native and click on 'Build'. This will generate the binary file 'program' which you can find in `.pio/build/native/`. Once in

this directory or when you copied the file to your current directory, launch the application with `./program`.

Additional arguments can be given to the program, which are listed as follows:

`-d DIRECTORY`: The directory to use as the virtual filesystem (VFS).

`-e`: Erase the virtual filesystem before use.

`-h MAC_ADDRESS`: The MAC address to assign to this virtual machine.

`-p TCP_PORT`: The local TCP port to use for running the Meshtastic API.

Usage with Docker

If you do not own a Linux machine, or you just want to separate things, you might want to run the application inside a docker container.

The Image

To build docker image, type

```
docker build -t meshtastic/device .
```

Usage

To run a container, type

```
docker run --rm -p 4403:4403 meshtastic/device
```

or, to get an interactive shell on the docker created container:

```
docker run -it -p 4403:4403 meshtastic/device bash
```

You might want to mount your local development folder: firmware

```
docker run -it --mount  
type=bind,source=/PathToMyProjects/Meshtastic/  
Meshtastic-device-mybranch,target=/Meshtastic-device-  
mybranch -p 4403:4403 meshtastic/device bash
```

Build the native application

Linux native application should be built inside the container. For this you must run container with interactive console "-it", as seen above.

First, some environment variables need to be set up with command:

```
. ~/platformio/pvenv/bin/activate
```

You also want to make some adjustments in the bin/build-all.sh to

conform the amd64 build:

```
sed -i 's/^BOARDS_ESP32.*$/BOARDS_ESP32=""/' bin/  
build-all.sh  
sed -i 's/^BOARDS_NRF52.*$/BOARDS_NRF52=""/' bin/  
build-all.sh  
sed -i 's/echo "Building SPIFFS.*$/exit/' bin/build-  
all.sh
```

You can build amd64 image with command

```
bin/build-all.sh
```

Executing the application interactively

The built binary file should be found under name `release/latest/bins/universal/meshtastic_linux_amd64`. If this is not the case, you can also use the direct program name: `.pio/build/native/program`

To use Python CLI against exposed TCP port 4403, type this in the host machine:

```
meshtastic --info --host localhost
```

While you can interact with it in the same way as a normal device using a client that supports TCP connections, the application has its limitations. For example, rebooting is not implemented, so for

some settings to apply you have to restart the application.

Stop the container

Run this to get the ID:

```
docker ps
```

Stop the container with command:

```
docker kill <id>
```

Tip: you can just use the first few characters of the ID in docker commands

Meshtastic Integrations

The Meshtastic ecosystem is highly extensible and allows easy integration with a number of existing software products and projects.

Current Meshtastic integrations:

CalTopo / SARTopo - Track Meshtastic nodes in CalTopo and SARTopo.

ATAK Plugin - Official Meshtastic ATAK Plugin for sending CoT to IMeshService in the Meshtastic Android app.

MQTT - Bridging mesh networks over the internet and integrating Meshtastic protocols with popular technologies such as Home Assistant, Node Red, and Adafruit IO.

Support for the integrated products should be sought from their respective authors or vendors.

CalTopo / SARTopo

CalTopo / SARTopo

Meshtastic can integrate with CalTopo Desktop edition quite easily through the product's APRS over serial support functionality.

Configuring the Meshtastic device

To configure our Meshtastic device for this integration, we have a couple of different options, both of which utilize the Serial module:

Enabling serial over the device's USB port

Serial over USB

```
meshtastic --set serial.enabled true --set  
serial.baud BAUD_9600 --set serial.mode CALTOPO --  
set serial.override_console_serial_port true
```

Enabling serial over an external USB to Serial adapter

External serial adapter

```
meshtastic --set serial.enabled true --set  
serial.baud BAUD_9600 --set serial.mode CALTOPO --  
set serial.txd = 13 --set serial.rxd = 14
```

ⓘ INFO

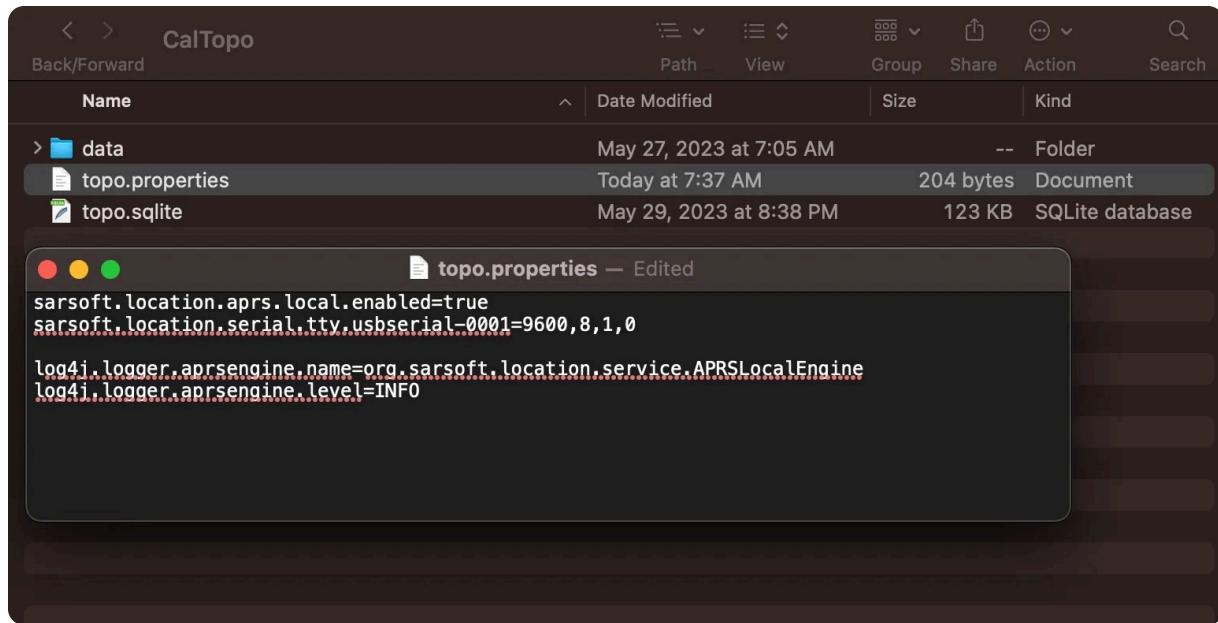
Ensure that serial baud rate is set to 9600 on both the Meshtastic device and the CalTopo / SARTopo topo.properties configuration

Setting up CalTopo / SARTopo

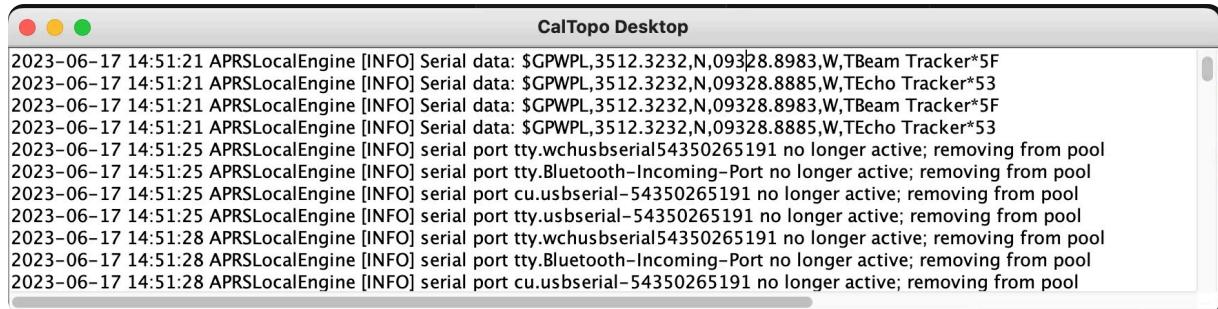
To setup CalTopo for Meshtastic integration using the Live Tracking via APRS, refer to the official documentation.

Example topo.properties file configuration for use with

Meshtastic:



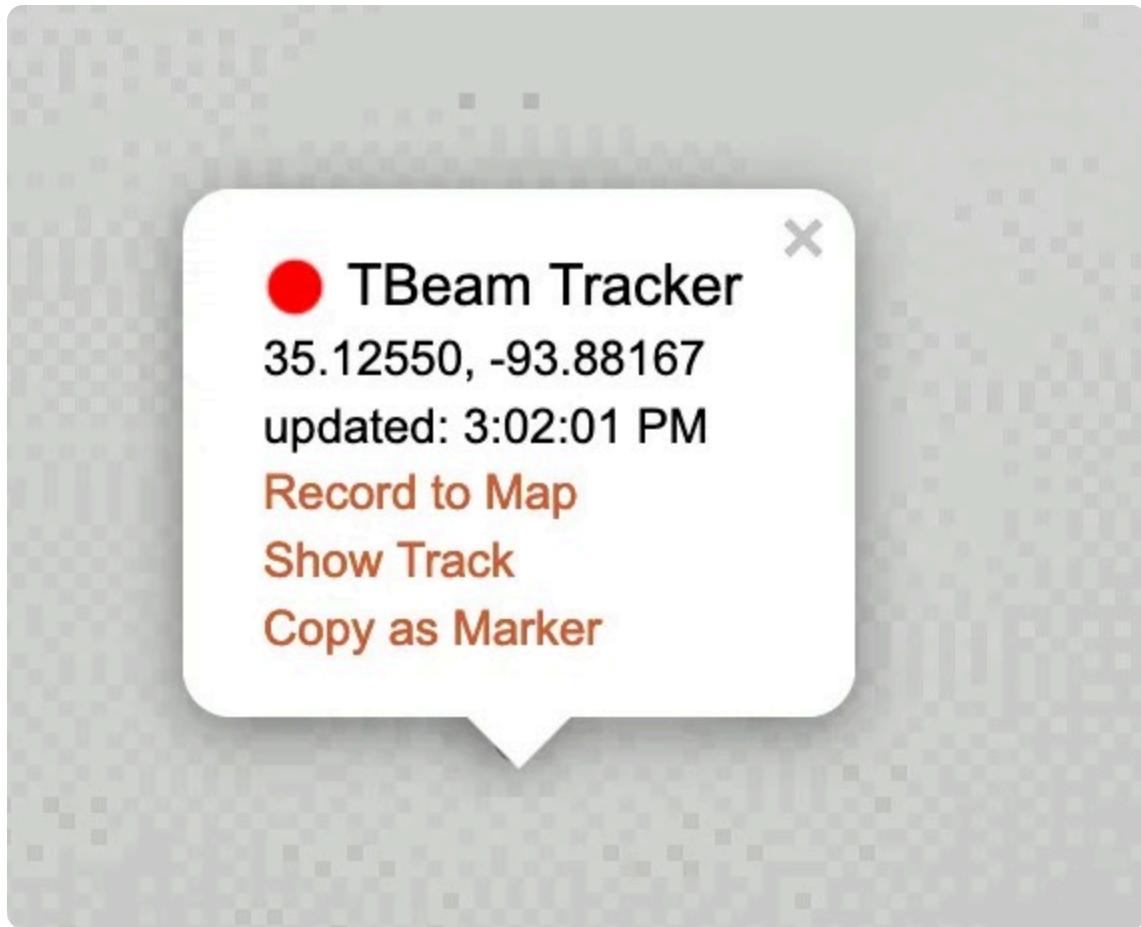
After starting CalTopo Desktop, NMEA waypoint location sentences should be streamed into the logs from the connected Meshtastic device every 10 seconds:



In the desktop's web UI for your CalTopo map, scroll down and check the **Shared Locations** checkbox under **Realtime Data**. Your nodes should appear on the map as points if they are connected correctly.



You can click on one or more of the node points and in the resulting tooltip, click **Record to Map**



In the resulting dialog, you can assign attributes such as a label or color to the live track created by the node.



When you view the shared map on another device or mobile, the

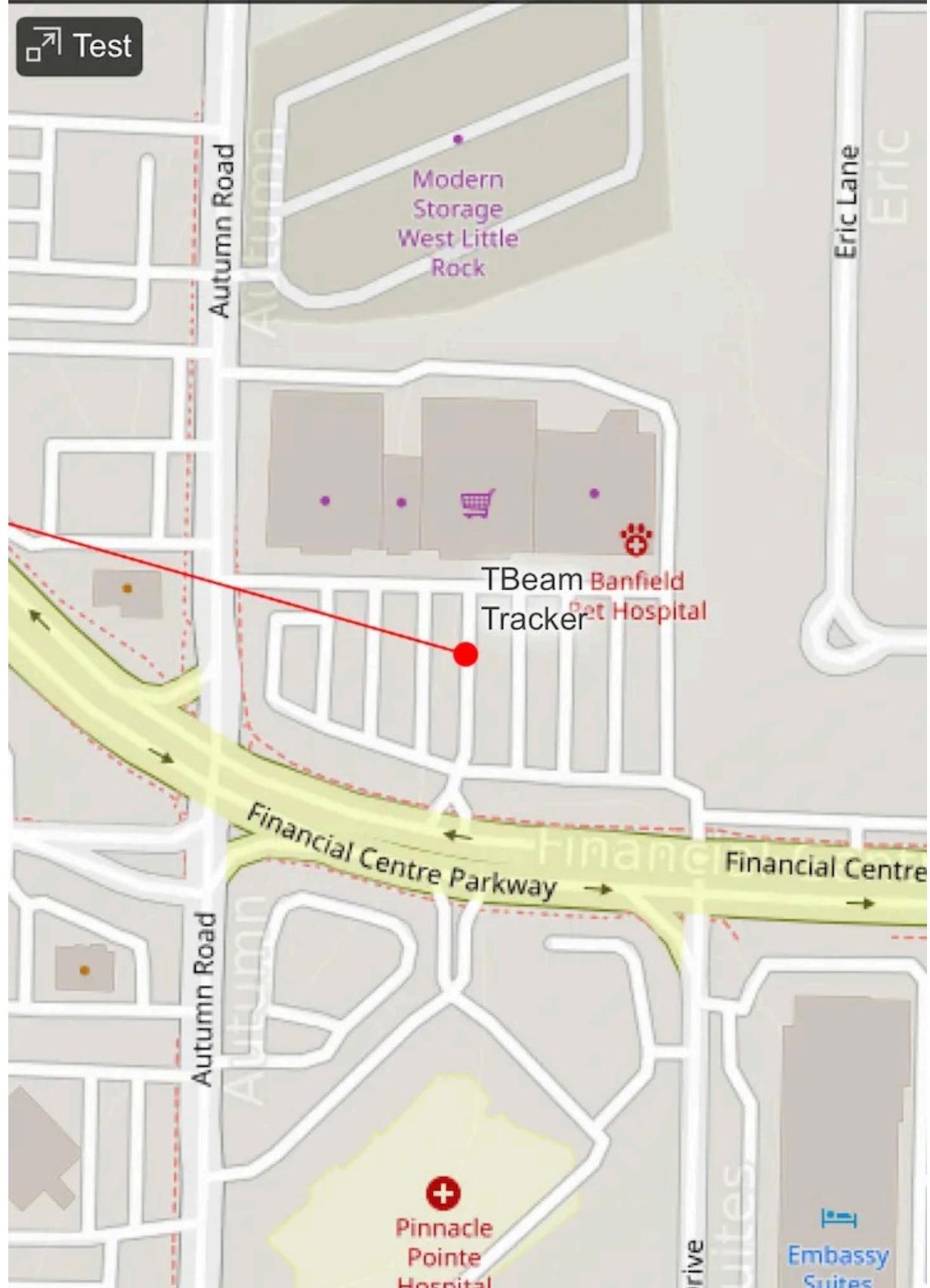
nodes should appear there as well now, if both the desktop and mobile device have internet connectivity.

1:35

5G



Test



ATAK Plugin

Official Meshtastic ATAK Plugin

Meshtastic can integrate with ATAK on Android using the Official ATAK Plugin.

Introduction

The ATAK plugin does not permit any Meshtastic configuration. The plugin does three things:

Binds to the IMeshService provided by the Meshtastic Android App in order to send.

Intercepts all outgoing ATAK CoT via the ATAK "PreSendProcessor" Interface and sends them to the IMeshService.

Listens for broadcasts from the Meshtastic Android App on the ATAK_PLUGIN portnum packets.

Instructions

Use the Meshtastic Android App on all parties' devices, and ensure they can talk to their local LoRa radio. Confirm they are able to achieve basic text messaging using the App.

Set the device's role to **TAK** in the device configuration settings.

Install the version of the Meshtastic ATAK-Plugin that matches the

ATAK version on all participants' devices from the project's GitHub Releases.

With the Meshtastic Android App running in the background (to ensure the IMeshService is alive), launch ATAK (with the Meshtastic ATAK-Plugin installed or install it once ATAK is running) and you should observe a green Meshtastic icon in the bottom right. If the icon is red, then the plugin was not able to bind to the IMeshService provided by the Meshtastic Android App. If this is the case, check to ensure the Meshtastic Android App is functioning. The plugin will reconnect after a failed bind without restarting ATAK. If you do not see a Meshtastic icon, make sure that you have installed the Meshtastic ATAK-Plugin correctly.

Standalone TAK Tracker usage

For devices with GPS available, configuring the device's role to `TAK_TRACKER` will allow the Meshtastic to transmit TAK PLI (Position Location Information) independently of ATAK. This data can be received and displayed within ATAK EUDs connected to a Meshtastic device and configured with the Meshtastic ATAK plugin (provided they are configured on the same Meshtastic channels).

A couple of important notes regarding this setup:

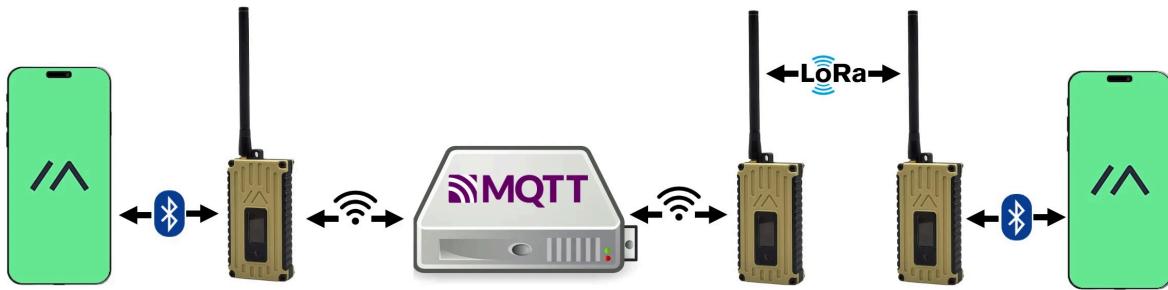
The intervals and behavior of TAK PLI broadcast will honor the settings configured for the standard Position configuration in

Meshtastic.

The callsign sent with the PLI will appear in ATAK with the User Long Name configured for the Meshtastic device.

MQTT | Integrations Overview

Bridging networks



Meshtastic networks in different locations beyond the reach of LoRa can be easily bridged together using MQTT. The simplest option is to connect your mesh to the official Meshtastic MQTT broker. This makes your devices appear on the world map, and provides a copy of your mesh traffic, translated into JSON. All you have to do to join the public MQTT server is to Enable MQTT and set Uplink and Downlink on the channels that you want to share over MQTT. The default device configuration using the public MQTT Server is encrypted.

You can also specify your own private MQTT broker to bridge mesh networks together, via the internet (or just a local IP network).

You can find the settings available for MQTT here.

IMPORTANT

When MQTT is turned on, you are potentially broadcasting your entire mesh's traffic onto the public internet. This includes messages and position information.

The default channel (LongFast) on the public MQTT server usually has a lot of traffic. Your device may get overloaded and may no longer function properly anymore. It is recommended to use a different channel or to use your own MQTT server if you experience issues.

Software Integrations

Using or emitting packets directly in/from smart home control software such as Home Assistant or other consumers that can work with JSON messages.

When MQTT is enabled, the Meshtastic device simply uplinks and/or downlinks every raw protobuf MeshPacket that it sees to the MQTT broker, encapsulated in a ServiceEnvelope protobuf. In addition, some packet types are serialized or deserialized from/to JSON messages for easier use in consumers. All packets are sent to the broker, whether they originate from another device on the mesh, or the gateway node itself.

MQTT Topics

If no specific root topic is configured, the default root topic will be `msh/`. Each device that is connected to MQTT will publish its MQTT state (`online`/`offline`) to:

`msh/2/stat/USERID`, where `USERID` is the node ID of the gateway device (the one connected to MQTT).

For each channel where uplink and/or downlink is enabled, two other topics might be used:

Protobufs topic

A gateway node will uplink and/or downlink raw (protobuf) MeshPackets to the topic:

`msh/2/e/CHANNELNAME/USERID`, where `CHANNELNAME` is the name of the channel (firmware versions prior to 2.3.0 will publish to a topic with `/c/` in the place of `/e/`).

For example: `msh/2/e/LongFast/!abcd1234`

The payload is a raw protobuf, whose definitions for Meshtastic can be found here. Reference guides for working with protobufs in several popular programming languages can be found [here](#).

Looking at the MQTT traffic with a program like `mosquitto_sub` will tell you it's working, but you won't get much useful information

out of it. For example:

```
◊?????"!  
!937bed1cTanksTnk"D???05??=???aP`  
ShortFast           !937bed1c
```

If `encryption_enabled` is set to true, the payload of the MeshPacket will remain encrypted with the key for the specified channel.

JSON topic

ⓘ NOTE

JSON is not supported on the nRF52 platform.

If JSON is enabled, packets from the following port numbers are serialized to JSON: `TEXT_MESSAGE_APP`, `TELEMETRY_APP`, `NODEINFO_APP`, `POSITION_APP`, `WAYPOINT_APP`, `NEIGHBORINFO_APP`, `TRACEROUTE_APP`, `DETECTION_SENSOR_APP`, `PAXCOUNTER_APP` and `REMOTE_HARDWARE_APP`. These are then forwarded to the topic:

```
msh/2/json/CHANNELNAME/USERID.
```

An example of a received `NODEINFO_APP` message:

```
{  
  "id": 452664778,
```

The meaning of these fields is as follows:

"`id`" is the unique ID for this message.

"`channel`" is the channel index this message was received on.

"`from`" is the unique node number of the node on the mesh that sent this message.

"`id`" inside the payload of a `NODEINFO_APP` message is the user ID of the node that sent it, which is currently just the hexadecimal representation of the node number.

"`hardware`" is the hardware model of the node sending the `NODEINFO_APP` message.

"`longname`" is the long name of the device that sent the `NODEINFO_APP` message.

"`shortname`" is the short name of the device that sent the `NODEINFO_APP` message.

"`sender`" is the user ID of the gateway device, which is in this case the same node that sent the `NODEINFO_APP` message (the hexadecimal value `7efeee00` represented by an integer in decimal is `2130636288`).

"`timestamp`" is the Unix Epoch when the message was received, represented as an integer in decimal.

"`to`" is the node number of the destination of the message. In this case, "-1" means it was a broadcast message (this is the decimal integer representation of `0xFFFFFFFF`).

"`type`" is the type of the message, in this case it was a

`NODEINFO_APP` message.

The `from` field can thus be used as a stable identifier for a specific node. Note that in firmware prior to 2.2.0, this is a signed value in JSON, while in firmware 2.2.0 and higher, the JSON values are unsigned.

If the message received contains valid JSON in the payload, the JSON is deserialized and added as a JSON object rather than a string containing the serialized JSON.

JSON downlink to instruct a node to send a message

You can also send a JSON message to the topic `msh/2/json/mqtt/` to instruct a gateway node to send a message to the mesh. Note that the channel you publish it on **must** be called "mqtt". The JSON message should contain the following fields:

```
{  
    "from": <node number of MQTT node>,  
    "to": <node number of recipient for a DM  
(optional)>,  
    "channel": <channel index (optional)>,  
    "type": "type",  
    "payload": {  
        "key": "value"  
        ...  
    }  
}
```

`from` and `payload` fields are required for a valid envelope (note that in firmware <2.2.20 a field `sender` was required, but this is no longer the case). The `from` field should be equal to the node number of the node that will transmit the message. Optionally, you can specify a different channel than the primary channel by setting the `channel` field to a channel index (0-7). Furthermore, you can send a direct message by setting the `to` field to the node number of the destination. If the `to` field is not set, the message will be broadcast to all nodes on the mesh.

Currently two types of messages are supported: `"sendtext"` and `"sendposition"`. For the type `sendtext`, the `payload` should be a string containing the text to send. For the type `sendposition`, the payload should be an object with the fields `latitude_i`, `longitude_i`, `altitude` (optional) and `time` (optional).

Basic Configuration

Check out MQTT Settings for full information. For quick start instructions, read on.

Connect your gateway node to wifi, by setting the `network.wifi_ssid`, `network.wifi_psk` and `network.wifi_enabled` preferences.

Alternatively use the RAK4631 with Ethernet Module RAK13800, by setting `network.eth_mode` and `network.eth_enabled` (note that

JSON is not supported on the nRF52 platform).

Configure your broker settings: `mqtt.address`, `mqtt.username`, and `mqtt.password`. If all are left blank, the device will connect to the Meshtastic broker.

Set `uplink_enabled` and `downlink_enabled` as appropriate for each channel. Most users will just have a single channel (at channel index 0). `meshtastic --ch-index 0 --ch-set uplink_enabled true`

`uplink_enabled` will tell the device to publish mesh packets to MQTT. `downlink_enabled` will tell the device to subscribe to MQTT, and forward any packets from there onto the mesh.

Gateway nodes

Any meshtastic node that has a direct connection to the internet (either via a helper app or installed WiFi/4G/satellite hardware) can function as a "Gateway node".

Gateway nodes (via code running in the phone) will contain two tables to whitelist particular traffic to either be delivered toward the internet, or down toward the mesh. Users that are developing custom apps will be able to customize these filters/subscriptions.

Since multiple gateway nodes might be connected to a single mesh, it is possible that duplicate messages will be published on any particular topic. Therefore, subscribers to these topics should

deduplicate if needed by using the packet ID of each message.

Optional web services

Public MQTT broker service

An existing public MQTT broker will be the default for this service, but clients can use any MQTT broker they choose.

Examples

Using mosquitto on a mac

Sending/receiving messages on mosquitto server using python

Using MQTT with Node-RED

Using MQTT with Home Assistant

Using MQTT with Adafruit IO

Mosquitto

Using mosquitto on a mac

install mqtt server

```
brew install mosquitto
```

start the mqtt server

```
brew services restart mosquitto
```

Do a quick test of server, start a subscriber on a topic:

Note: this will wait until you press control-c (publish a message, see below)

```
mosquitto_sub -t test/hello
```

In another window, publish a message to that topic:

```
mosquitto_pub -h localhost -q 0 -t test/hello -m  
'yo!'
```

For Meshtastic to be able to access that server, two settings need to be changed in the `/usr/local/etc/mosquitto/mosquitto.conf` file:

```
listener 1883 0.0.0.0
```

Restart the service:

```
brew services restart mosquitto
```

If you are using the mac firewall, you will need to go into: System Preferences > Security & Privacy > Firewall > Firewall Options and add it.

Python

Sending/receiving messages on mosquitto server using python

Here is an example publish message in python (run `pip install paho-mqtt` first):

```
#!/usr/bin/env python3
import paho.mqtt.client as mqtt
from random import uniform
import time

client =
mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.connect('localhost')

while True:
    randNumber = uniform(20.0, 21.0)
    client.publish("env/test/TEMPERATURE",
randNumber)
    print("Just published " + str(randNumber) + " to
topic TEMPERATURE")
    time.sleep(1)
```

Here is example subscribe in python:

```
#!/usr/bin/env python3
```


Node-RED

Using MQTT with Node-RED

Node-RED is a free cross-platform programming tool for wiring together hardware, APIs, and online services developed originally by IBM for IOT. It is widely used for home automation by many non-professional programmers and runs well on Pi's. Node-RED has many plug-in modules written by the community.

I will use this platform as a practical example on how to interface with the MQTT features of Meshtastic. Everything can be done from GUI's without using command line.

Enabling MQTT

Use <http://client.meshtastic.org/>, the python CLI, or an Apple or Android app to connect to your device and adjust these settings.

Settings--> Radio Config--> Network

On the node that will act as the gateway between the mesh and MQTT enable a network connection (i.e. Wifi, Ethernet).

Save

Settings--> Module Config--> MQTT config

Configure the MQTT gateway's network configuration.

Verify Encryption Enabled is OFF.

(optional) Turn JSON Output Enabled ON.

Save

Channel Editor

Go to Channel Editor and enable Uplink and Downlink on the channels you wish to publish to MQTT.

Save

Using Node-RED with Meshtastic

There are three common approaches:

Using JSON-encoded messages

Using protobuf-encoded messages with the Meshtastic decode node

Using protobuf-encoded messages with a protobuf decode node and the Meshtastic protobuf definitions

The JSON output only publishes the following subset of the messages on a Meshtastic network:

Text Message

Telemetry

Device Metrics

Environment Metrics

Power Metrics

Node Info

Position

Waypoint

Neighbor Info

JSON is intended to be a convenience for consuming data in other applications like Home Assistant.
Protobufs are mesh native.

1. Using JSON-encoded messages

 **NOTE**

JSON is not supported on the nRF52 platform.

Make sure that option *JSON Output Enabled* is set in MQTT module options and you have a channel called "mqtt".

Below is a valid JSON envelope for information sent by MQTT to a device for broadcast onto the mesh. The `to` field is optional and can be omitted for broadcast. The `channel` field is also optional and can be omitted to send to the primary channel.

```
{  
    "from": <node number of the transmitter>,  
    "to": <node number of the receiver for a DM  
(optional)>,  
    "channel": <channel index (optional)>,  
    "type": "sendtext",  
    "payload": text or a json object go here  
}
```

2. Using protobuf-encoded messages with the Meshtastic decode node

Install Node-Red plug-in: <https://flows.nodered.org/node/@meshtastic/node-red-contrib-meshtastic>

More info is in the plug-in source repository.

There is an example flow using this mechanism available
<https://github.com/scruplelesswizard/meshtastic-node-red>

3. Using protobuf-encoded messages with a protobuf decode node and the Meshtastic protobuf definitions

If you don't want to depend on JSON decoding on the device, you can decode the protobuf messages off-device. To do that you will need to get the .proto files from <https://github.com/meshtastic/protobufs>. They function as a schema and are required for decoding in Node-RED. Save the files where the node-RED application can access them and note the file path of the "mqtt.proto" file.

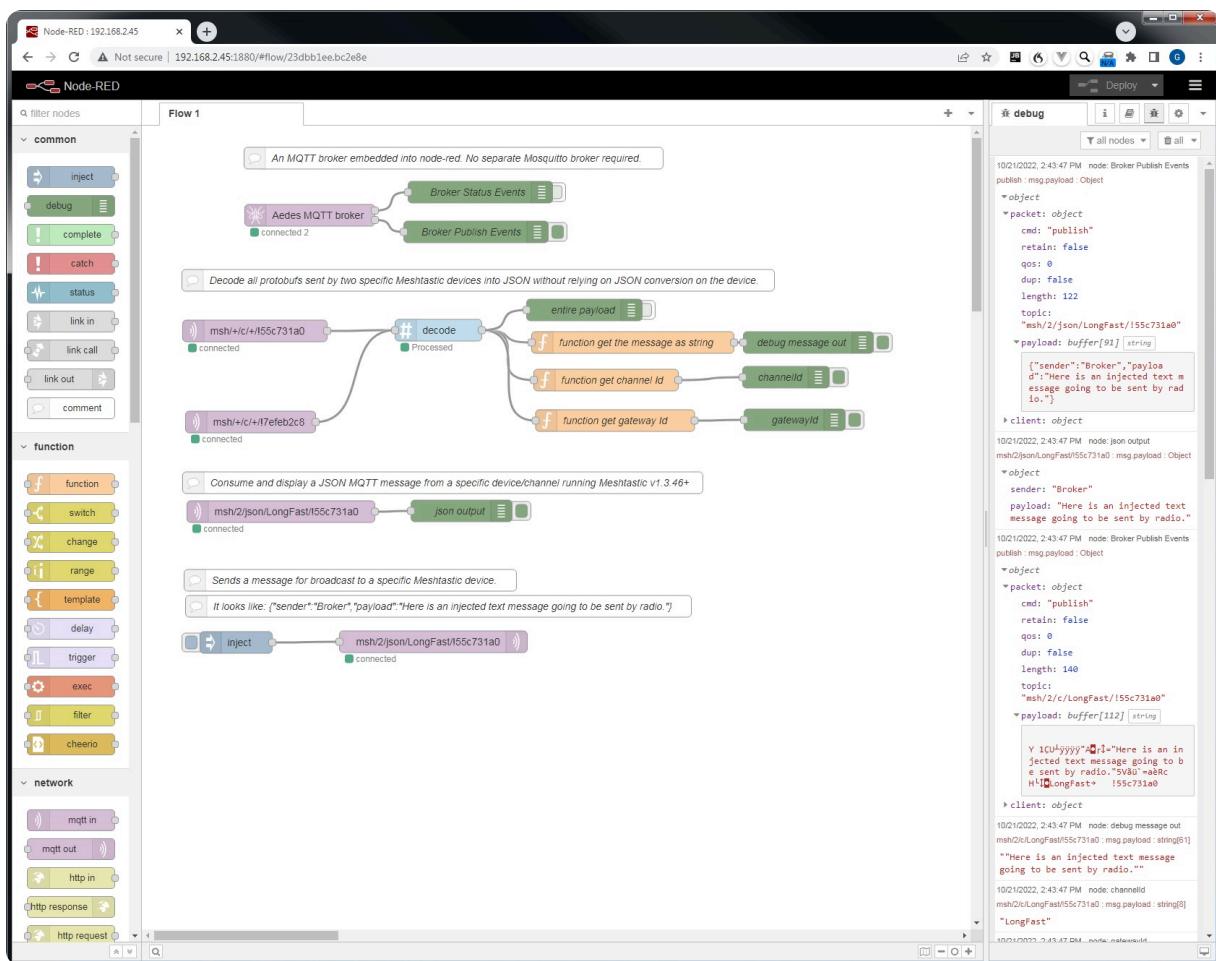
Install Node-RED plug-ins to your Node-RED application for an embedded MQTT server and a protobuf decoder.

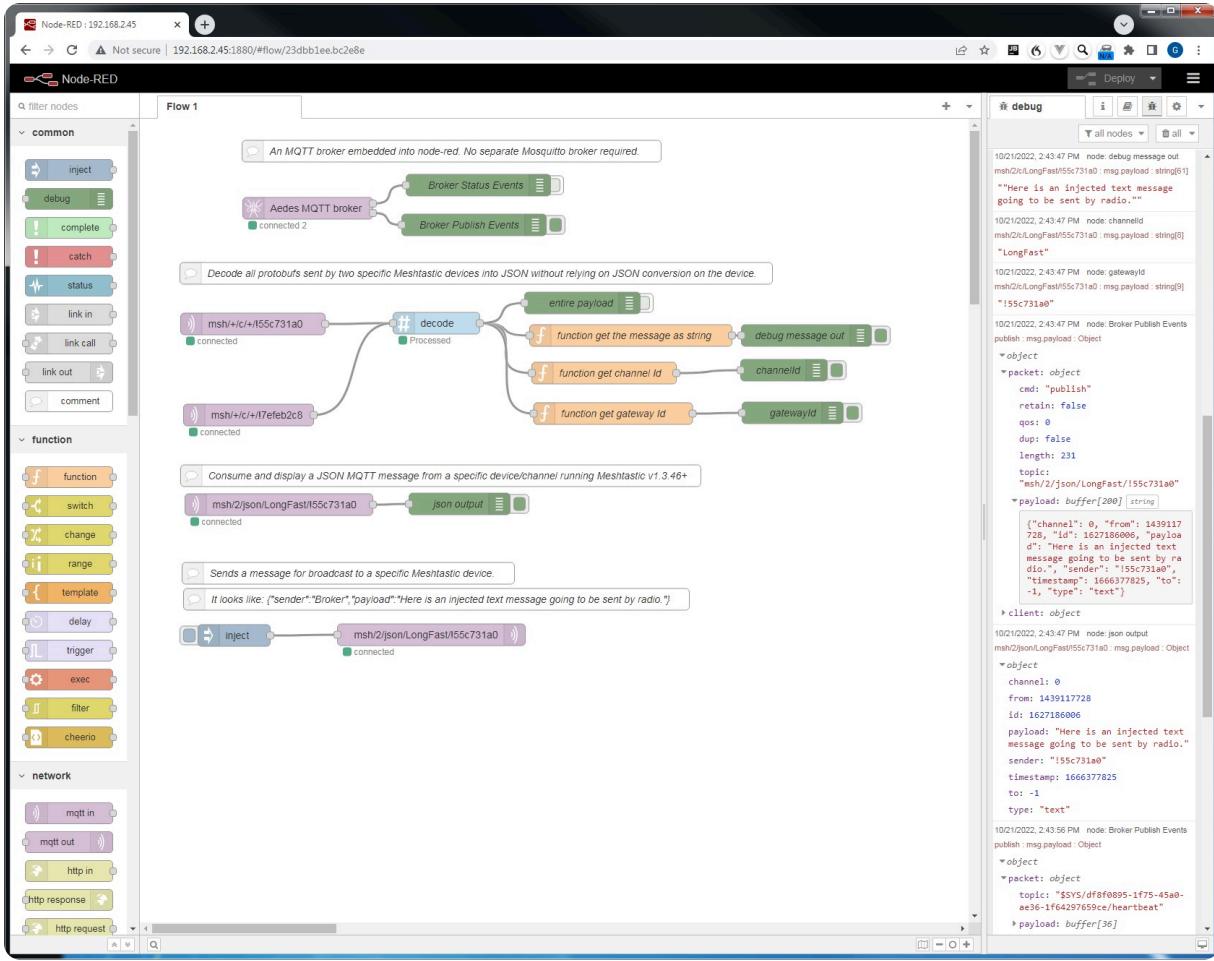
<https://flows.nodered.org/node/node-red-contrib-aedes>

<https://flows.nodered.org/node/node-red-contrib-protobuf>

Drag, drop, and wire the nodes like this. For this example, I ran Node-RED on a Windows machine. Note that file paths might be

specified differently on different platforms. MQTT server wild cards are usually the same. A "+" is a single level wildcard for a specific topic level. A "#" is a multiple level wildcard that can be used at the end of a topic filter. The debug messages shown are what happens when the inject button sends a JSON message with a topic designed to be picked up by the specified Meshtastic device and then having it rebroadcast the message.





The screenshot displays five configuration panels for nodes in a Node-RED flow:

- Edit aedes broker node**: Configuration for an aedes broker node. It includes fields for Name (Name), Connection (Security), MQTT port (1883), WS bind (port), WS port (Enter Websocket port. Leave blank to disable Websocket support), Enable secure (SSL/TLS) connection, and DB Url (mongodb://localhost:27017/mqtt).
- Edit debug node**: Configuration for a debug node. It includes fields for Output (msg.payload), To (checkboxes for debug window, system console, and node status (32 characters)), and Name (Broker Publish Events).
- Edit mqtt in node**: Configuration for an mqtt in node. It includes fields for Server (mqtt ip ad:1883), Action (Subscribe to single topic), Topic (msh/+/_/l55c731a0), QoS (2), Output (a Buffer), and Name (Name).
- Edit decode node**: Configuration for a decode node. It includes fields for Name (Name), Proto File (E:\Meshtastic-protobufs-master\mqtt.proto), and Type (ServiceEnvelope).
- Edit function node**: Configuration for a function node. It includes tabs for Properties, Setup, On Start, On Message, and On Stop. The On Message tab contains the following JavaScript code:

```

1 msg.payload = msg.payload.packet.decoded.payload;
2 
3 let bufferObj = Buffer.from(msg.payload, "base64");
4 let decodedString = bufferObj.toString("utf8");
5 msg.payload = decodedString;
6 
7 return msg;

```

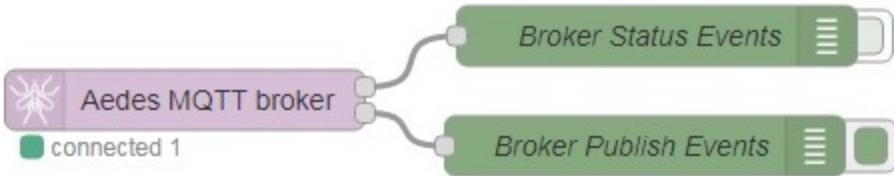
- Edit function node**: Configuration for another function node. It includes tabs for Properties, Setup, On Start, On Message, and On Stop. The On Message tab contains the following JavaScript code:

```

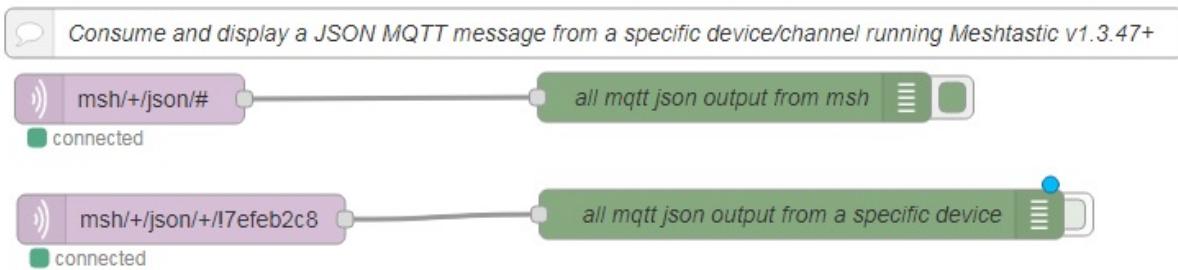
1 msg.payload = msg.payload.channelId;
2 
3 return msg;

```

The aedes broker must be set up on the same flow as the other nodes. By activating the Publish debug node, you can see all the published messages.



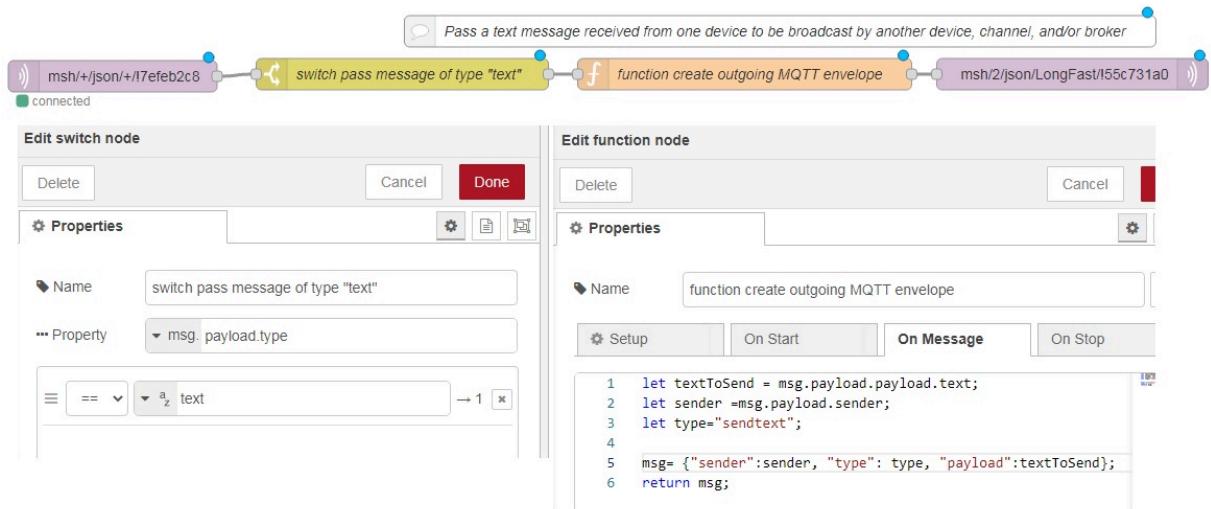
Receiving a json mqtt message is very simple.



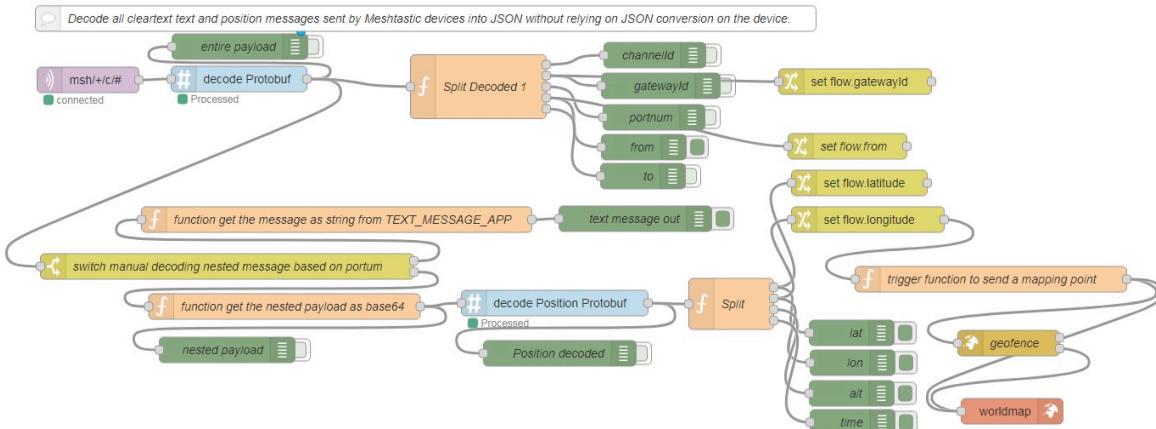
Injecting a json message to be sent by a device is also very simple. You do need the correct envelope.



Forwarding a text message from one device, through a broker, to another broker/device/channel would look like this.



If you want to decode text and position messages without json, it gets complicated:

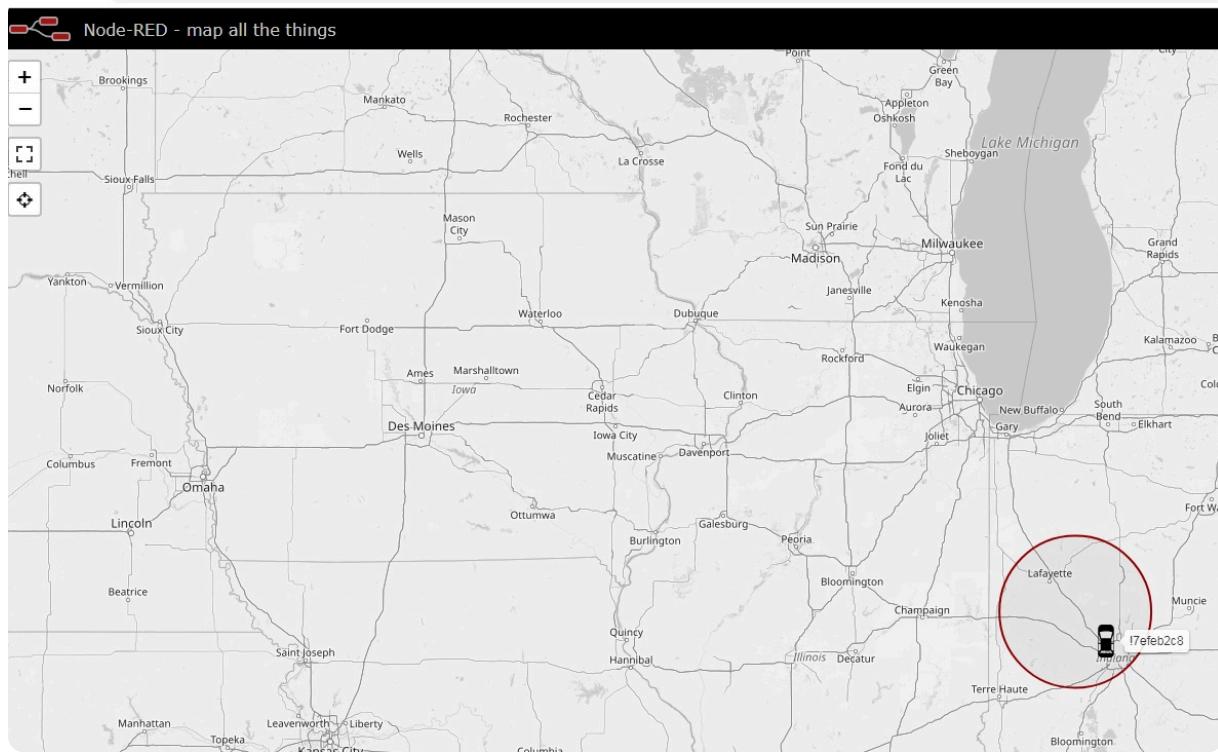


If you are interested in my flow for this it is here:

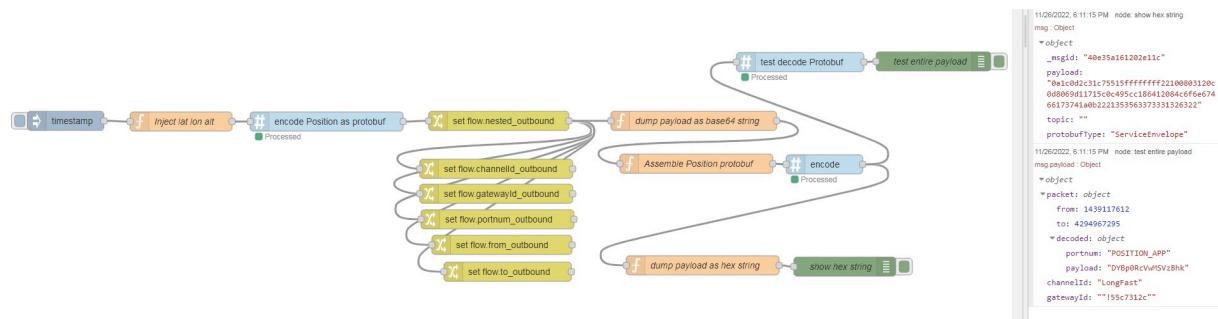
```
[
{
  "id": "10fe1b2e9cb3feb2",
  "type": "decode",
  "z": "23dbb1ee.bc2e8e",
  "name": "decode Protobuf",
  "protofile": "a0d4288141f6a629",
```

(documents/mqtt/Flow.txt)

Node-red can rapidly (minutes vs days) put together some pretty impressive output when paired with meshtastic. Here is the output of that flow geofencing and mapping via mqtt data.



Advanced use, such as encoding Position and sending it to a device via MQTT without using JSON can get a little complicated. An example of how it can be done is below.



The flow is:

```
[  
  {  
    "id": "32ca608d9e7c5236",  
    "type": "inject",  
    "z": "23dbb1ee.bc2e8e",  
    "name": "",  
    "props": [{ "p": "payload" }, { "p": "topic", "vt": "str" }],  
    "repeat": "",  
    "crontab": "",  
    "once": false,  
    "onceDelay": 0.1,  
    "topic": "",  
    "payload": "",  
    "payloadType": "date",  
    "x": 96.5,  
    "y": 1952,  
    "wires": [[{"id": "2b536512e8c7aef2"}]]  
,  
  {  
    "id": "20bbd2d1408b8dc5",  
    "type": "change",  
    "z": "23dbb1ee.bc2e8e",  
    "name": "",  
    "rules": [  
      {  
        "t": "set",  
        "p": "channelId_outbound",  
        "pt": "flow",  
        "to": "LongFast",  
        "tot": "str"  
      }  
    ]  
  }  
]
```

Sending a position to a device for broadcast to the mesh is much easier with JSON. It requires the message to be published to a channel called "mqtt". You can let the message send out to a different channel by setting the `channel` field to a channel index (0-7). Then use the MQTT Service Envelope type: "sendposition". A valid MQTT envelope and message to broadcast lat, lon, altitude (optional) and time (optional) looks like this:

```
{  
  "from": 2130636288,  
  "type": "sendposition",  
  "payload": {  
    "latitude_i": 399600000,  
    "longitude_i": -862600000,  
    "altitude": 100,  
    "time": 1670201543  
  }  
}
```

An example of doing this in node-red:



Edit function node

Delete

Properties

Name: injection JSON Position

Setup On Start **On Message**

```
1 msg.payload = {  
2     "sender": "Broker",  
3     "type": "sendposition",  
4     "payload": {  
5         "latitude_i": 399600000,  
6         "longitude_i": -862600000,  
7         "altitude": 100,  
8         //make unix time from javascript timestamp  
9         "time": Math.trunc(msg.payload / 1000)  
10    }  
11 }  
12 return msg;
```

```
node: all mqtt json output from msh  
msh/2/json/LongFast/!7efec08c : msg.payload : Object  
  object  
    channel: 0  
    from: 2130624652  
    id: 471436283  
  payload: object  
    altitude: 100  
    latitude_i: 399600000  
    longitude_i: -862600000  
    time: 1670200279  
    sender: "!7efec08c"  
    timestamp: 0  
    to: -1  
    type: "position"
```


Home Assistant

Home Assistant Integrations for Meshtastic

 **NOTE**

Due to a known issue with nRF52 devices and JSON, nRF52 devices are not supported for use with Home Assistant at this time.

Integrating Meshtastic into Home Assistant brings a new level of control and monitoring to your mesh network. On this page, we'll guide you through the process of creating Meshtastic MQTT sensor entities within Home Assistant. Whether you want to keep an eye on battery levels, environmental conditions, or even receive notifications from your mesh network, these integrations provide you with the tools to make it happen.

 **INFO**

It is highly recommended to download MQTT Explorer for analyzing the JSON threads that come across the broker. This can be downloaded here <http://mqtt-explorer.com/>.

Create Meshtastic MQTT Sensor Entities

Ensure your mesh unit is connected to your MQTT broker and using JSON as an output.

Open configuration.yaml and include the following line:

```
mqtt: !include mqtt.yaml
```

Create a new text file called `mqtt.yaml` in the root directory of home assistant (the same folder as configuration.yaml).

Copy the following code to the mqtt.yaml file.

Default Node Telemetry Example

```
sensor:  
  
# Node #1  
  
- name: "Node 1 Battery Voltage"  
  unique_id: "node_1__battery_voltage"  
  state_topic: "msh/2/json/LongFast/!67ea9400"  
  state_class: measurement  
  value_template: >-  
    {% if value_json.from == 4038675309 and
```

Telemetry Module Entities

If you have environmental sensors installed, create entities for these by including some or all of the following blocks:

```
- name: "Node 1 Temperature"
  unique_id: "node_1_temperature"
  state_topic: "msh/2/json/LongFast/!67ea9400"
  state_class: measurement
  value_template: >-
    {% if value_json.from == 4038675309 and
value_json.payload.temperature is defined %}
      {{ (((value_json.payload.temperature | float)
* 1.8) +32) | round(2) }}
    {% else %}
      {{ this.state }}
    {% endif %}
  device_class: "temperature"
  unit_of_measurement: "F"
# With device_class: "temperature" set, make sure to
use the configured unit for temperature in your HA
instance.
# If you don't, then non-temperature messages will
change the value of this sensor by reinterpreting the
current state with the wrong unit, unless you account
for it.
# For Celsius use:    {{
(value_json.payload.temperature | float) | round(1)
```

Message Entities

For added functionality you can create an entity for messages received from a node:

```
- name: "Node 1 Messages"
  unique_id: "node_1_messages"
  state_topic: "msh/2/json/LongFast/!67ea9400"
  value_template: >-
    {% if value_json.from == 4038675309 and
value_json.payload.text is defined %}
      {{ value_json.payload.text }}
    {% else %}
      {{ this.state }}
    {% endif %}
```

Additional Entities

Home Assistant entities can be created for any data type that is published to MQTT. For example: altitude, latitude_i, longitude_i, time, current, and neighbors. Use the templates above as a guide to create additional entities if desired.

Configure With Your Topic & Node ID's

In every entity, replace `msh/2/json/LongFast/!67ea9400` with the topic your node publishes to. In this example, `!67ea9400`

refers to the node that has mqtt enabled on the mesh and is publishing to the broker.

In every entity replace `4038675309` with the node number of the radio you wish to monitor. In this example `4038675309` is the node on the mesh with environment sensors and telemetry that I wish to observe. Node numbers can be found by monitoring the output in MQTT Explorer, listening with the MQTT addon or by using the Python CLI with `meshtastic --info`.

Additional Nodes

Copy and paste these entities then change `name`, `unique_id`, `from`, and `states` to create entities from any additional nodes' parameters:

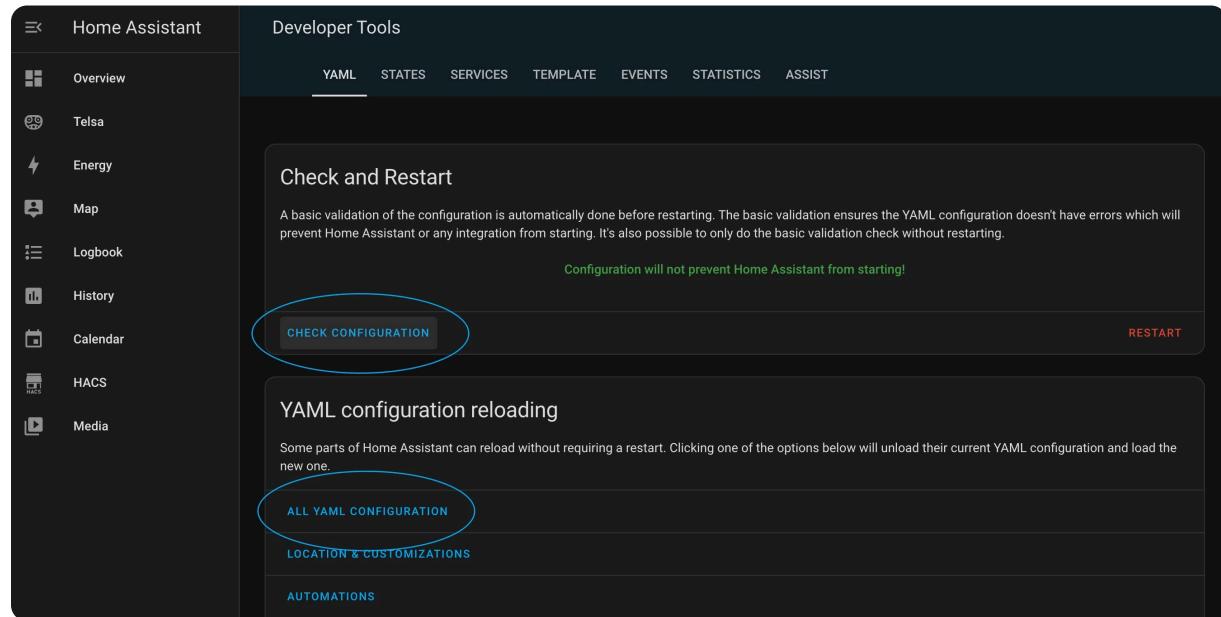
```
- name: "Node 2 Messages"
  unique_id: "node_2_messages"
  state_topic: "msh/2/json/LongFast/!67ea9400"
  value_template: >-
    {% if value_json.from == 695318008 and
    value_json.payload.text is defined %}
      {{ value_json.payload.text }}
    {% else %}
      {{ this.state }}
    {% endif %}
```

Check Configuration & Reload YAML

In Home Assistant, run `CHECK CONFIGURATION` in the developer tools section and then reload all yaml configuration.

🔥 WARNING

Always Check Configuration before reloading YAML or restarting Home Assistant to identify errors.



Create Dashboard Cards

Entities Card for Telemetry

Create a new Entities Card and select the entities you have created.

Entities Card Configuration

?

Title (optional)
Meshtastic Nodes

Theme (optional)

Show Header Toggle? Color icons based on state?

Header: None

Footer: None

Entities (required)

- Entity Node 1 ChUtil
- Entity Node 1 AirUtilTX
- Entity Node 1 Battery Voltage
- Entity Node 1 Battery Percent
- Entity Node 1 Temperature
- Entity Node 1 Humidity
- Entity Node 1 Pressure
- Entity Node 1 Gas Resistance
- Entity

[SHOW CODE EDITOR](#)

[CANCEL](#) [SAVE](#)

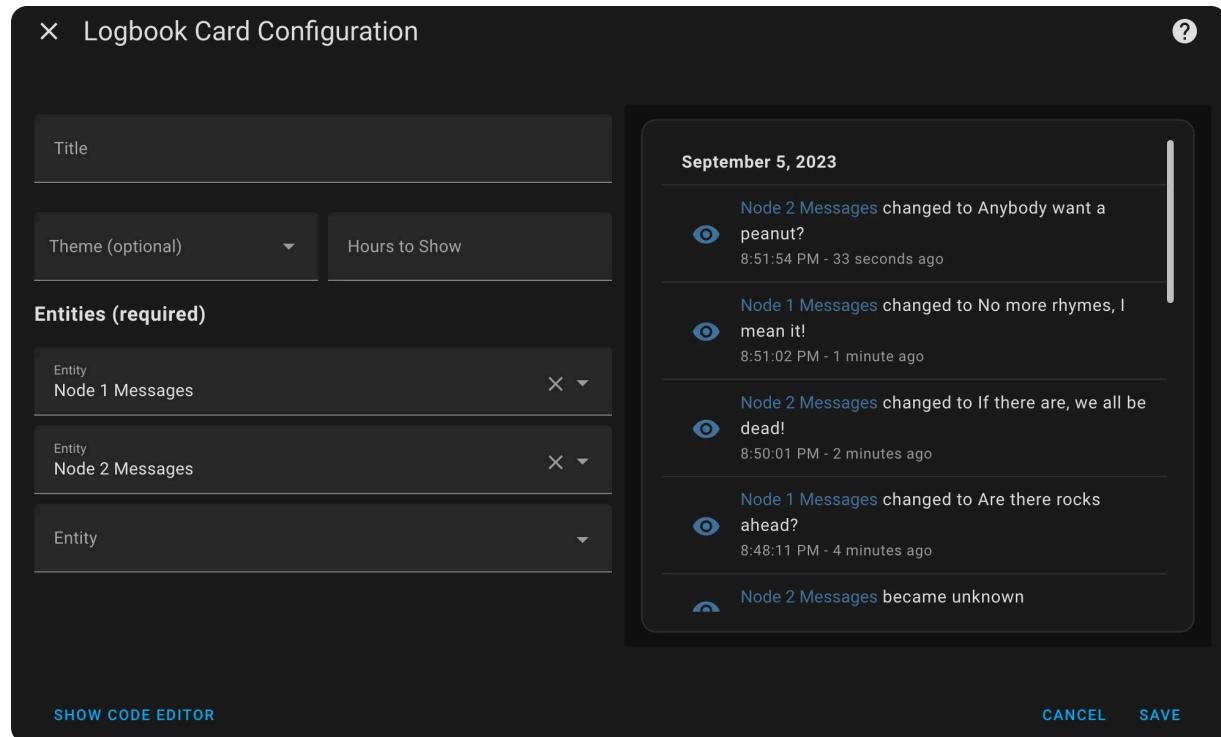
Meshtastic Nodes

 Node 1 ChUtil	27.67%
 Node 1 AirUtilTX	0.8%
 Node 1 Battery Voltage	4.28 Volts
 Node 1 Battery Percent	101.0%
 Node 1 Temperature	75.83 F
 Node 1 Humidity	48.3%
 Node 1 Pressure	994.26 hPa
 Node 1 Gas Resistance	92.66 MOhms

Logbook Card for Messaging

The logbook card is useful to keep a record of incoming messages from the mesh. Below is an example of how the logbook card

would be set up.

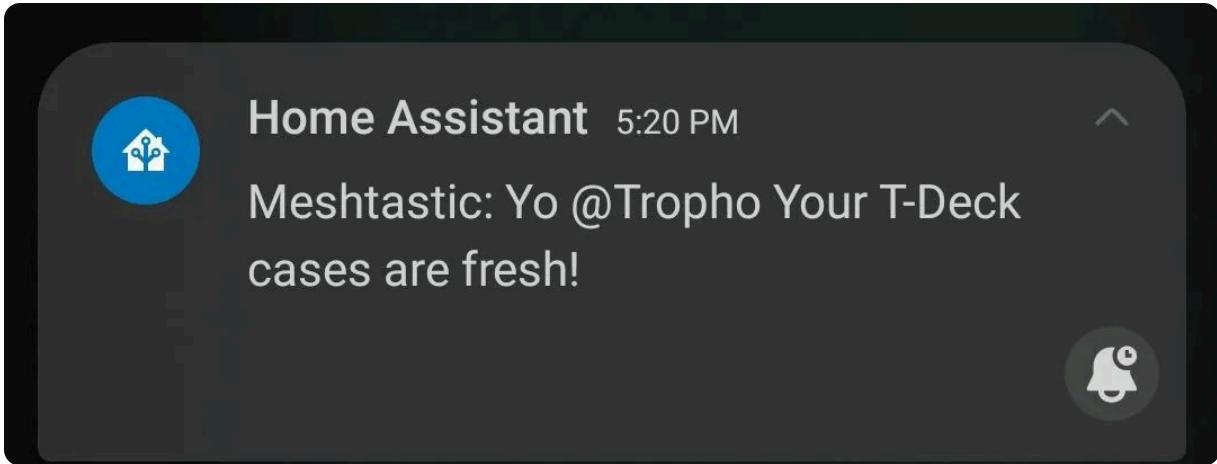


Trigger HA Automations

It is possible to have Home Assistant trigger automations based on messages or events on your mesh.

Notifications

This example waits for a message containing @Tropho and then sends a pop-up notification to his flip phone with the message. Optionally you can have ALL messages from the mesh sent as HA notifications to your phone.



Add the following code to your `automations.yaml` file. Be sure to modify the `topic`, `regex_search`, and `service` for your configuration.

```
- alias: Meshtastic - New notification
  description: any message with an @Tropho will send to mobile device
  trigger:
    - platform: mqtt
      topic: msh/2/json/LongFast/!67ea9400
  condition:
    - condition: template
      value_template: "{{trigger.payload_json.payload.text | regex_search('@Tropho') }}"
      # or send ALL messages from the mesh to HA notifications (except for range tests)
      # value_template: "{{trigger.payload_json.payload.text is defined and "seq" not in trigger.payload_json.payload.text}}"
```

This same type of automation is very useful to trigger other actions in Home Assistant. For example, you could turn on a fan when the temperature reaches a certain value, or a play a sound on a speaker when a new message is received.

Create a Send Message Entity

It is possible to create an input text box to send messages to your mesh from within Home Assistant.

Input Text Helper Entity

First, create an input text helper entity. The preferred way to configure an input text is via the HA interface at **Settings > Devices & Services > Helpers**. Click the add button and then choose the **Text** option. Make a text input helper with a max length of 190 to be on the safe side. See example below:

Create Text

X

Name*

Meshtastic Send Box



Icon

mdi:chat

X

▼

Minimum length

0

Maximum length

190

Display mode



Text



Password

Regex pattern

BACK

CREATE

Create a Send Message Automation

This automation will check the send box for changes. After typing a message, either hit enter or click off the box and the automation

will send a text string in JSON to the mqtt broker. Make sure to publish to a channel called "mqtt" and to update the device ID and `from` field in the example below. A field `channel` can be added to transmit on a different channel index than the primary, or a `to` field can be added with a node number to send a DM.

```
- alias: Meshtastic - Send Automation
  description: ""
  trigger:
    - platform: state
      entity_id:
        - input_text.meshtastic_send_box
  condition: []
  action:
    - delay:
        hours: 0
        minutes: 0
        seconds: 1
        milliseconds: 0
    - service: mqtt.publish
      data:
        qos: 0
        retain: false
        topic: msh/2/json/mqtt/!67ea9400
        payload: >-
          {"from":1743426560,"type":"sendtext","payload":"
            {{states('input_text.meshtastic_send_box')}}"}
    - delay:
```

Add this card to your dashboard by going to Edit Dashboard -> + ADD CARD. Then search BY ENTITY for Meshtastic Send Box and check the box next to the entry. Click CONTINUE, then ADD TO DASHBOARD.

Adafruit IO

Adafruit IO for Meshtastic

Adafruit IO can be used to graph telemetry and messages from a Meshtastic network via json/mqtt. The following example script will listen for node packets and publish voltage, rssi, snr and messages to individual feeds on adafruit IO. If a feed doesn't exist it will be created. Be aware, however, that the free Adafruit account is limited to 10 feeds. Once your feeds are being populated with data you can create dashboards to display graphs and gauges with the data.

 **INFO**

To utilize this script you need to have an Adafruit IO account, a working mqtt broker setup and a mesh node that is publishing to the mqtt broker.

You will need to modify the ini sample file with your Adafruit IO and mqtt credentials Code repository is here [https://bitbucket.org/tavdog/mesh_aio_logger/src/main/]

```
[GENERAL]
PRINT_CONFIG = false
; https://pynative.com/list-all-timezones-in-python/
TIMEZONE = US/Hawaii
```

```
# Persistent mqtt client, watch for voltage, telemetry, message p  
and publish them to io  
  
# Import Adafruit IO REST client.  
from Adafruit_IO import Client, Feed, Data, RequestError  
from datetime import datetime  
import paho.mqtt.client as mqtt  
import os  
import sys  
import json  
import time  
import pytz  
import configparser  
  
config = configparser.ConfigParser()  
# Read the configuration file  
config.read('config.ini')  
  
if config.getboolean('GENERAL', 'PRINT_CONFIG'):  
    # Iterate through sections and options to print all config va  
    for section in config.sections():  
        print(f"[{section}]")  
        for key, value in config.items(section):  
            print(f'{key} = {value}')  
        print() # Add an empty line between sections for better  
readability  
  
# set your timezone here  
TIMEZONE = config['GENERAL']['TIMEZONE']  
CHANNEL_LIST = config['GENERAL']['CHANNEL_LIST']  
MQTT_SERVER = config['MQTT']['SERVER']
```


Meshtastic Community



Local Groups

Below is a list of groups that have been actively organizing Meshtastic networks i...



Community Apps

2 items



Enclosures

5 items

Local Groups

Below is a list of groups that have been actively organizing Meshtastic networks in their regions. Feel free to contact them for assistance in getting started or if you're interested in contributing to the network. If you're a group organizer with an online presence and wish to be included in this list, please edit this page directly or reach out to us on Discord to add your group.

Canada

Alberta

YYC Mesh

United States

Arkansas

Fort Smith Mesh

California

SoCal Mesh

Laguna Mesh

Mission Viejo Mesh

Bay Area Mesh
San Diego Mesh
Antelope Valley Mesh

Colorado

Denver Mesh

Hawaii

Hawaii Meshnet

Massachusetts

Boston Meshnet

Texas

Austin Mesh

United Kingdom

UK Meshtastic Kent / South East

The Netherlands

Meshtastic Netherlands

Community Apps

The Meshtastic ecosystem is highly extensible, and a number of community projects have been made to fit different people's needs. If you wish to create your own application or module, please read the information in the developers section, and tell us about your project on the forum.

Current community projects:

Meshtasticator (Simulator) - Meshtasticator is a discrete-event and interactive simulator that mimics the radio section of the device software.

Meshtastic Web API - Meshtastic Web API provides a RESTful interface to interact with a Meshtastic node via a serial connection.

Support for these projects should be sought from their respective authors.

Meshtasticator

 **NOTE**

This is a community project maintained by @GUVWAF.

Development can be followed on GitHub. Support should be sought from the respective authors.

Meshtasticator is a discrete-event and interactive simulator that mimics the radio section of the device software and can be used to assess the performance of your scenario, or the scalability of the protocol. Meshtasticator was created and is maintained by GUVWAF and more information on its use and setup can be found on the Meshtasticator Github page

Meshtastic Web API

 **NOTE**

This is a community project maintained by @bmswens.

Development can be followed on GitHub. Support should be sought from the respective authors.

Overview

Meshtastic Web API provides a RESTful interface to interact with a Meshtastic node via a serial connection.

Links

[Source](#)

[API Docs](#)

[Installation Instructions](#)

[Supported Services](#)

[Mattermost](#)

Meshtastic Community Enclosures



RAK WisBlock

1 items



T-Beam V1

Enclosures



T-Echo

Enclosures



LILYGO® Lora V2

Enclosures



Heltec V2

Enclosures

RAK WisBlock Enclosures

**Created by KeithMon/Voltaic
Enclosures**

Solar Base Station

Download from Printables or purchase from the creator's Etsy Store.



Created by tropho/TonyG

RAK5005 Case

Download from Printables or purchase from the creator's Etsy Store.



RAK19007 Case

Download from Printables or purchase from the creator's Etsy Store.



RAK19003 Case

Download from Printables or purchase from the creator's Etsy Store.



RAK19003 (Micro) Case

Download from Printables or purchase from the creator's Etsy Store.



**Created by tavdog/Tavis
Gustafson**

Harbor Breeze Meshtastic Hack



RAK WisBlock Harbor Breeze Solar Light Enclosure Hack

**Created by tavdog/Tavis
Gustafson**

**Low Cost Harbor Breeze Solar Light RAK
Enclosure Hack**



The Harbor Breeze Solar LED light can be purchased for about \$15 and includes a capable solar panel, lithium ion cell and charge controller in a waterproof enclosure. The RAK baseboards fit perfectly inside the solar compartment. Below are the steps for completing this hack for a completely weatherproof solar powered Meshtastic node.

Skills needed

Some soldering is required as well as drilling a hole in the plastic case.

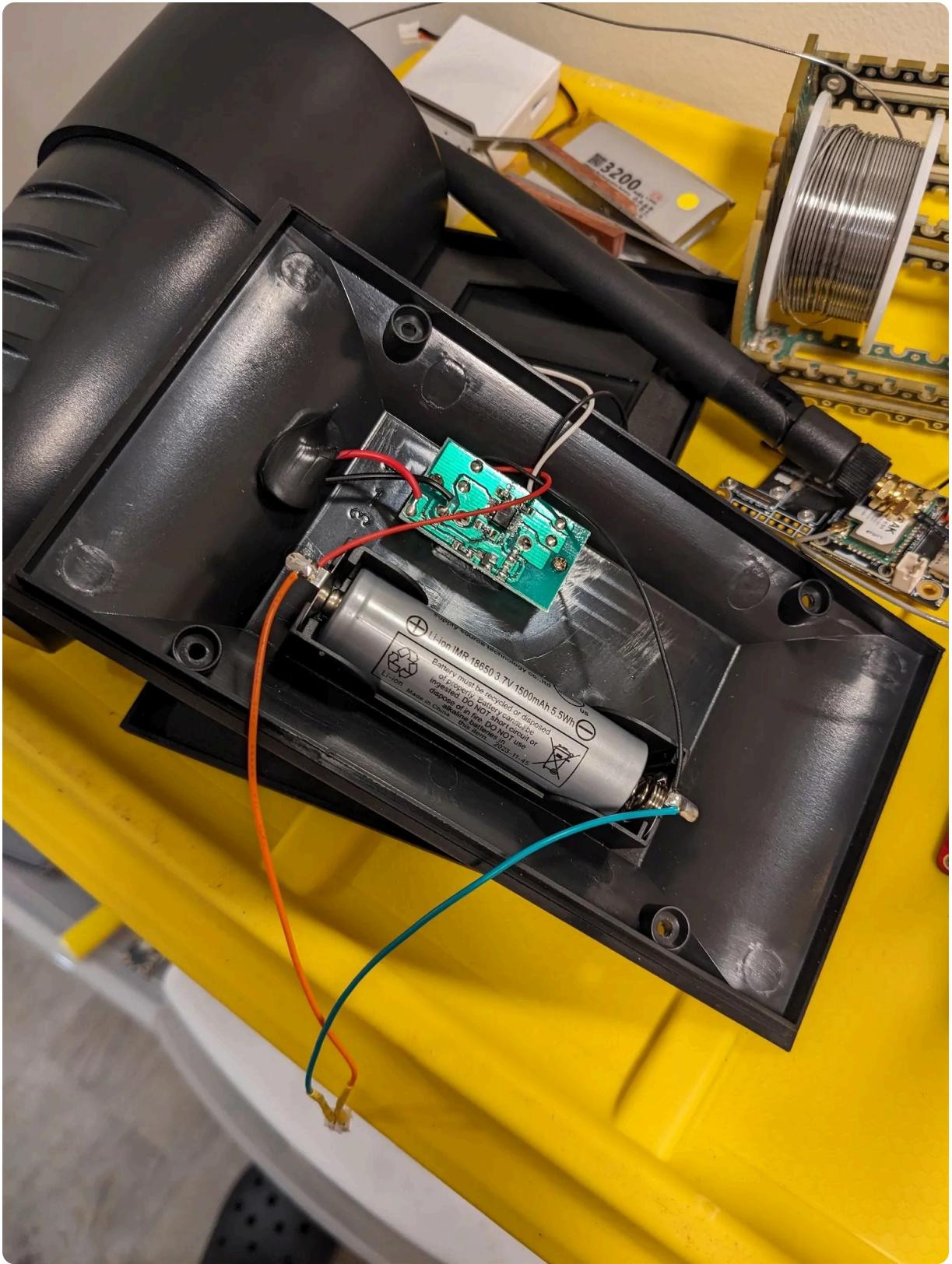
Items needed

An ipex to sma pigtail is needed if in order to have an external antenna. A JST-PHR-2 connector is required if you want a plug-in type battery connection. Soldered connection will also work though. Sealing glue (silicone or marine sealant)

Steps

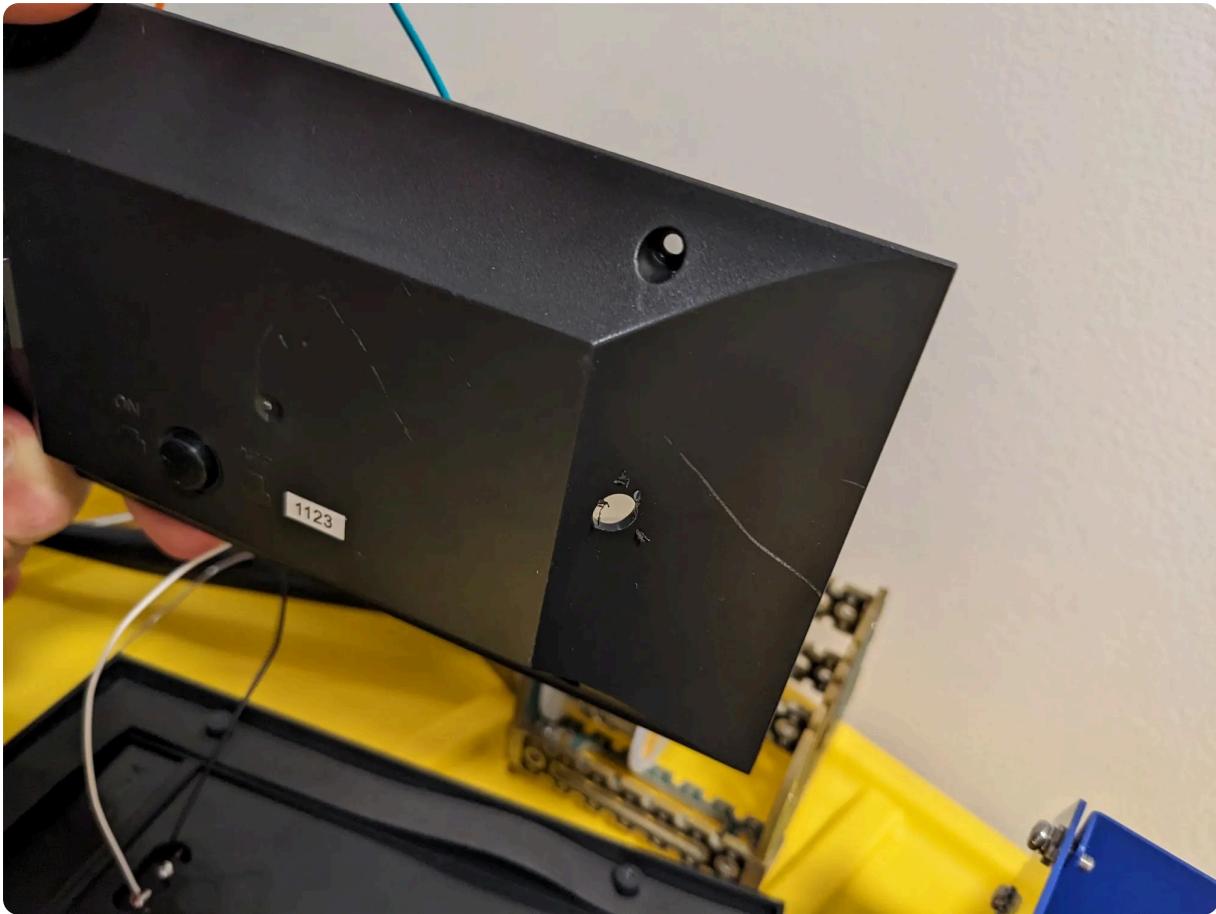
Step 1. Unscrew 4 small screws to open the solar compartment.

Step 2. Solder your battery JST-PHR-2 connector or power wires to the + and - terminals of the battery. If you don't have the proper connector you can solder directly to the pins on the back of the JST connector, just make sure you get the polarity correct.



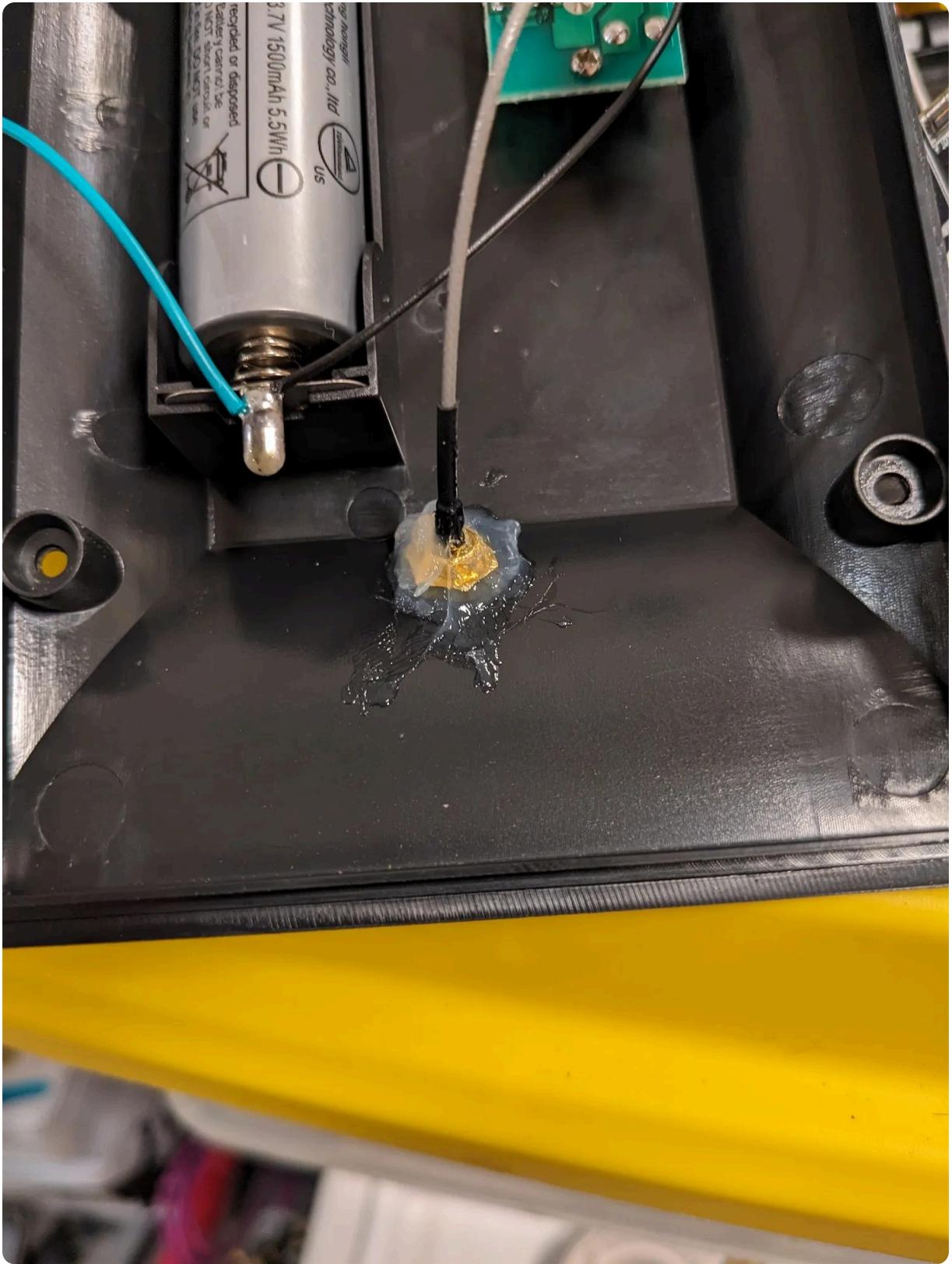
Step 3. Drill the hole for your sma pigtail. A perfectly vertical antenna via 90 degree bend will not allow for a very flat solar panel so choose wisely or use an externally mounted antenna. You can skip this step if you want to use an internal antenna that fits.



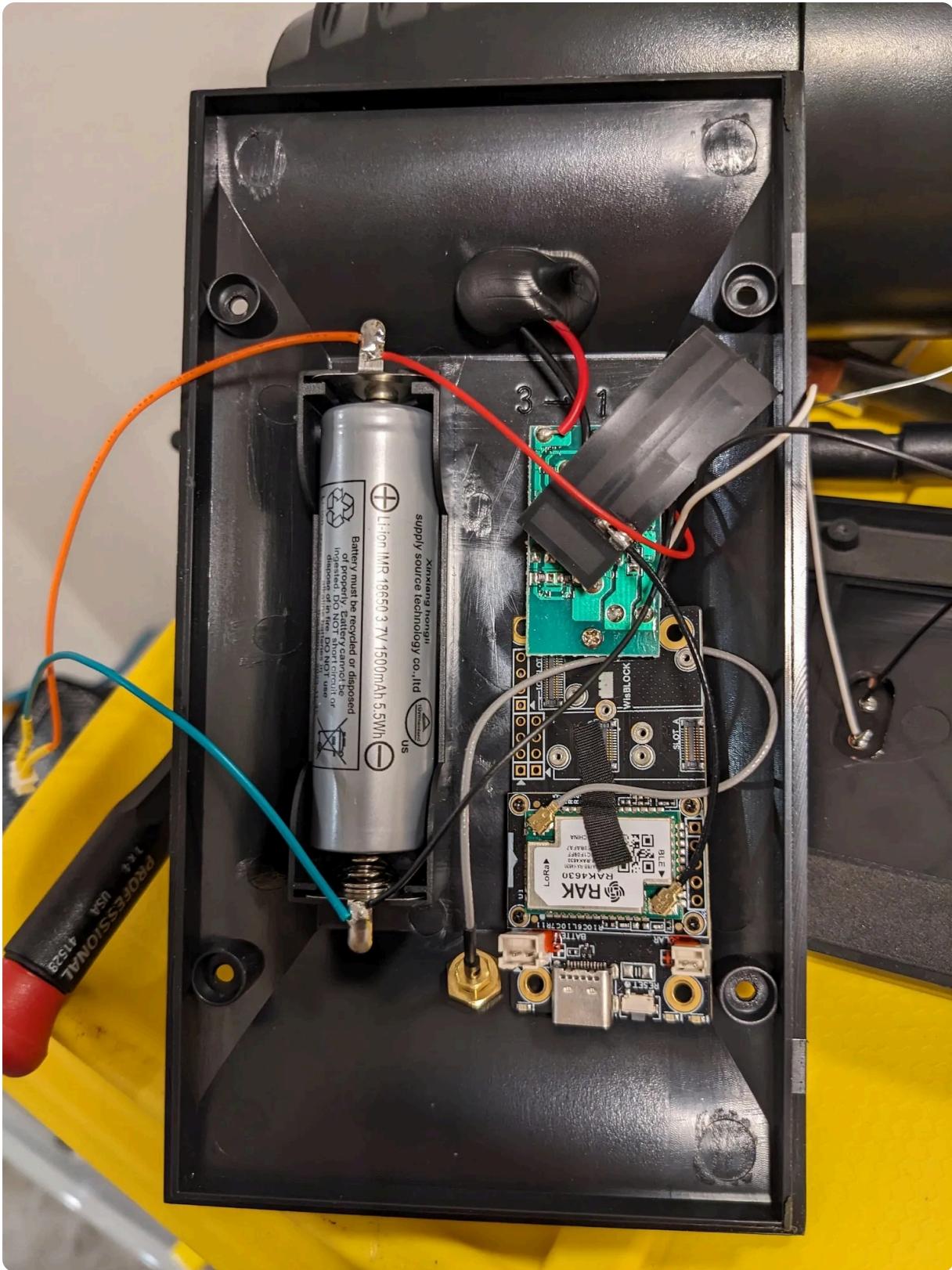


Step 4. Install the sma pigtail. Seal with silicone or marine sealant.





Step 5. Install the RAK module in the free space. You can lightly tack glue this in place if you want to.



Step 6. Connect the antenna and battery and close up the back.

The light housing can be removed if not needed.



The light housing is not as waterproof as the solar enclosure so be careful if putting electronics inside of it. Attach to a stick and put it high up.



The light will still turn on at night for a very stealthy node. To turn

off the light there is a button on the back.

Links

Here is a Lowes.com link to the solar light :

(<https://www.lowes.com/pd/60LM-Solar-Spot-Light/1002689960>)

Here is a link to the Meshtastic Starter kit :

(<https://store.rakwireless.com/products/wisblock-meshtastic-starter-kit>)

Here is an amazon link to the sma and antennas

(<https://www.amazon.com/DIYmalls-915MHz>)

T-Beam Enclosures

Created by tropho/TonyG

T-Beam V5 Case

Download from Printables or purchase from the creator's Etsy Store.

Required Hardware

- (x4) M3x16mm socket-head cap screws
- (x4) M3 nuts
- (x4) M2x4mm screws (no nuts) to secure T-Beam to the frame



T-Echo Enclosures

Created by BrianN

T-Echo Expedition Case

Download from Thingiverse.



LILYGO® TTGO Lora Enclosures

Created by tropho/TonyG

**TTGO LoRa32 v2.1.1.6 Case (will also fit the
SX1280/v2.1.1.8 variety)**

Download from Printables or purchase from the creator's Etsy Store.

Required Hardware

- (x4) M3x16mm socket-head cap screws
- (x4) M3 Nuts
- (x2) M2 Screws (to secure TTGO LoRa32 board to frame)
- (x1) LiPo battery pack (1,000mAh)
- (x1) Momentary micro push button for PRG (6x6x5mm)



Heltec LoRa 32 Enclosures

Created by tropho/TonyG

Heltec LoRa32 v2.1+ Case

Download from Printables or purchase from the creator's Etsy Store.

Required Hardware

- (x4) M3x20mm socket-head cap screws
- (x4) M3 Nuts
- (x1) LiPo battery pack (1,000mAh)
- (x1) Latching micro push button switch



Developers



Android

Build instructions



Device

4 items



Firmware

4 items



Web Client

Overview



Python

2 items

 **Javascript**

3 items

 **Docs**

3 items

 **Reference Material**

4 items

Building the Android App

Build instructions

If you would like to develop this application we'd love your help! These build instructions are brief and should be improved, please send a PR if you can.

Use Android Studio to build/debug

Use `git submodule update --init --recursive` to pull in the various sub-modules we depend on.

There are a few config files which you'll need to copy from templates included in the project. Run the following commands to do so:

```
rm ./app/google-services.json  
cp ./app/google-services-example.json ./app/google-  
services.json  
rm ./app/src/main/res/values/curfirmwareversion.xml  
cp ./app/special/curfirmwareversion.xml ./app/src/  
main/res/values/
```

Now you should be able to select "Run / Run" in the IDE and it will happily start running on your phone or the emulator.

 **NOTE**

The emulators don't support Bluetooth, so some features can not be used in that environment.

Setup Analytics

Analytics are included but can be disabled by the user on the settings screen.

Configure analytics for development device

```
adb shell setprop debug.firebaseio.analytics.app  
com.geeksville.mesh  
adb shell setprop log.tag.FirebaseCrashlytics DEBUG
```

Set verbose logging

```
adb shell setprop log.tag.FA VERBOSE
```

Publish to Google Play

 **INFO**

Only available for core developers that publish releases.

Add repository secrets:

KEYSTORE_FILENAME

Name of the `.jks`

KEYSTORE

Convert the `.jks` to base64:

```
openssl base64 < filename.jks | tr -d '\n' | tee  
filename.txt
```

KEYSTORE_PROPERTIES

`storePassword=nononononono`

`keyPassword=nononononono`

`keyAlias=upload`

`storeFile=nononononono.jks`

Update protobufs

Go to Actions / Make Release / Run Workflow

Pick the Releases branch

Enter the version found in `app/gradle.build`

Device



Client API

This document describes the protocol for external API clients using our devices. If ...



HTTP API

This is a mini-spec on a HTTP API which can be used by browser based clients to i...



Module API

The purpose of this tutorial is for writing new core modules that can be run on a d...



Error Codes

The device might report these fault codes on the screen, but it will also be outputt...

Client API (Serial/TCP/ BLE)

This document describes the protocol for external API clients using our devices. If you are interested in running your own code on the device itself, see the module API documentation instead.

The Device API is designed to have only a simple stream of ToRadio and FromRadio packets and all polymorphism comes from the flexible set of Google Protocol Buffers which are sent over the wire. We use protocol buffers extensively both for the Bluetooth API and for packets inside the mesh or when providing packets to other applications on the phone.

Streaming version

This protocol is **almost** identical when it is deployed over BLE, Serial/USB, or TCP (our three currently supported transports for connecting to phone/PC). Most of this document is in terms of the original BLE version, but this section describes the small changes when this API is exposed over a Streaming (non datagram) transport. The streaming version has the following changes:

We assume the stream is reliable (though the protocol will re-synchronize if bytes are lost or corrupted). i.e. we do not include

CRCs or error correction codes.

Packets always have a four byte header (described below) prefixed before each packet. This header provides framing characters and length.

The stream going towards the radio is only a series of ToRadio packets (with the extra 4 byte headers)

The stream going towards the PC is a stream of FromRadio packets (with the 4 byte headers), or if the receiver state machine does not see valid header bytes it can (optionally) print those bytes as the debug console from the radio. This allows the device to emit regular serial debugging messages (which can be understood by a terminal program) but also switch to a more structured set of protobufs once it sees that the PC client has sent a protobuf towards it.

The 4 byte header is constructed to both provide framing and to not look like 'normal' 7 bit ASCII.

Byte 0: START1 (0x94)

Byte 1: START2 (0xc3)

Byte 2: MSB of protobuf length

Byte 3: LSB of protobuf length

The receiver will validate length and if >512 it will assume the packet is corrupted and return to looking for START1. While looking for START1 any other characters are printed as "debug output". For a small example implementation of this reader see the python

implementation.

Bluetooth (MeshBluetoothService)

This is the main Bluetooth service for the device and provides the API your app should use to get information about the mesh, send packets, or provision the radio.

For a reference implementation of a client that uses this service see `RadioInterfaceService`.

Typical flow when a phone connects to the device should be the following (if you want to watch this flow from the python app just run `meshtastic --debug --info` - the flow over BLE is identical):

There are only three relevant endpoints (and they have built in BLE documentation - so use a BLE tool of your choice to watch them): `FromRadio`, `FromNum` (sends notifies when new data is available in `FromRadio`) and `ToRadio`.

SetMTU size to 512.

Write a `ToRadio.startConfig` protobuf to the "ToRadio" endpoint - this tells the radio you are a new connection and you need the entire NodeDB sent down.

Read repeatedly from the "FromRadio" endpoint. Each time you read you will get back a `FromRadio` protobuf (see Meshtastic-protobuf). Keep reading from this endpoint until you get back an

empty buffer.

See below for the expected sequence for your initial download.

After the initial download, you should subscribe for BLE "notify" on the "FromNum" endpoint. If a notification arrives, that means there are now one or more FromRadio packets waiting inside FromRadio.

Read from FromRadio until you get back an empty packet.

Any time you want to send packets to the radio, you should write a ToRadio packet into ToRadio.

Expected sequence for initial download:

After your send startConfig, you will receive a series of FromRadio packets. The sequence of these packets will be as follows (but you are best not counting on this, instead just update your model for whatever packet you receive - based on looking at the type).

Read a RadioConfig from "radio" - used to get the channel and radio settings.

Read a User from "user" - to get the username for this node.

Read a MyNodeInfo from "mynode" to get information about this local device.

Read a series of NodeInfo packets to build the phone's copy of the current NodeDB for the mesh.

Read an endConfig packet that indicates the entire state you need has been sent.

Read a series of MeshPackets until it returns empty to get any messages that arrived for this node while the phone was away.

For definitions (and documentation) on FromRadio, ToRadio, MyNodeInfo, NodeInfo and User protocol buffers see mesh.proto

UUID for the service: 6ba1b218-15a8-461f-9fa8-5dcae273eafd

Each characteristic is listed as follows:

UUID Properties Description (including human readable name)

2c55e69e-4993-11ed-b878-0242ac120002 read fromradio - contains a newly received FromRadio packet destined towards the phone (up to MAXPACKET bytes per packet). After reading the ESP32 will put the next packet in this mailbox. If the FIFO is empty it will put an empty packet in this mailbox.

f75c76d2-129e-4dad-a1dd-7866124401e7 write toradio - write ToRadio protobufs to this characteristic to send them (up to MAXPACKET len)

ed9da18c-a800-4f66-a670-aa7547e34453 read,notify,write fromnum - the current packet # in the message waiting inside fromradio, if the phone sees this notify it should read messages until it catches up with this number.

The phone can write to this register to go backwards up to FIXME packets, to handle the rare case of a fromradio packet was dropped after the ESP32 callback was called, but before it arrives at the phone. If the phone writes to this register the ESP32 will discard older packets and put the next packet \geq fromnum in

fromradio. When the ESP32 advances fromnum, it will delay doing the notify by 100ms, in the hopes that the notify will never actually need to be sent if the phone is already pulling from fromradio.

Note: that if the phone ever sees this number decrease, it means the ESP32 has rebooted.

Re: Queue management, not all messages are kept in the fromradio queue (filtered based on SubPacket):

Only the most recent Position and User messages for a particular node are kept.

All Data SubPackets are kept.

No WantNodeNum / DenyNodeNum messages are kept.

A variable keepAllPackets, if set to true will suppress this behavior and instead keep everything for forwarding to the phone (for debugging).

A Note on MTU Sizes

This device will work with any MTU size, but it is highly recommended that you call your phone's "setMTU function to increase MTU to 512 bytes" as soon as you connect to a service. This will dramatically improve performance when reading/writing packets.

Protobuf API

On connect, you should send a want_config_id protobuf to the device. This will cause the device to send its node DB and radio config via the fromradio endpoint. After sending the full DB, the radio will send a want_config_id to indicate it is done sending the configuration.

Other Bluetooth Services

This document focuses on the core device protocol, but it is worth noting that the following other Bluetooth services are also provided by the device.

BluetoothSoftwareUpdate

The software update service. For a sample function that performs a software update using this API see startUpdate.

SoftwareUpdateService UUID cb0b9a0b-a84c-4c0d-bdbb-442e3144ee30

Characteristics

UUID	properties	description
e74dd9c0-a301-4a6f-95a1-f0e1dbea8e1e	write,read	total image

UUID	properties	description
		size, 32 bit, write this first, then read back to see if it was acceptable (0 mean not accepted)
e272ebac-d463-4b98-bc84-5cc1a39ee517	write	data, variable sized, recommended 512 bytes, write one for each block of file
4826129c-c22a-43a3-b066-ce8f0d5bacc6	write	crc32, write last - writing this will complete the OTA operation,

UUID	properties	description
		now you can read result
5e134862-7411-4424-ac4a-210937432c77	read,notify	result code, readable but will notify when the OTA operation completes
5e134862-7411-4424-ac4a-210937432c67	write	sets the region for programming, currently only 0 (app) or 100 (spiffs) are defined, if not set app is assumed
GATT_UUID_SW_VERSION_STR/0x2a28	read	We also implement these standard

UUID	properties	description
		GATT entries because SW update probably needs them:
GATT_UUID_MANU_NAME/0x2a29	read	
GATT_UUID_HW_VERSION_STR/0x2a27	read	

DeviceInformationService

Implements the standard BLE contract for this service (has software version, hardware model, serial number, etc...).

BatteryLevelService

Implements the standard BLE contract service, provides battery level in a way that most client devices should automatically understand (i.e. it should show in the Bluetooth devices screen automatically).

HTTP API

ⓘ **INFO**

This is a mini-spec on a HTTP API which can be used by browser based clients to interact with Meshtastic devices.

Why protobufs

No need for JSON parsing on the resource constrained embedded server.

Small.

Already in use for all other transports (so shared testing/tooling coverage).

Backwards and forward compatible.

Request headers

`Content-Type: application/x-protobuf`

Indicates protobuf content (Meshtastic protobufs)

Response headers

`Content-Type: application/x-protobuf`

Indicates protobuf content (Meshtastic protobufs)

X-Protobuf-Schema: <URI to the .proto schema file>

Not required but recommended for documentation/reflection purposes

Endpoints

Two endpoints are specified:

/api/v1/toradio

Allows `PUT` and `OPTION` requests.

`PUT`

A `PUT` request to this endpoint will be expected to contain a series of ToRadio protobuf payloads.

The protobufs will be sent in binary as the body for the request.

Only one ToRadio message per request is supported.

`OPTIONS`

An `OPTIONS` request to this endpoint will return a response status code `204` and headers only.

/api/v1/fromradio

Allows `GET` requests.

GET

A `GET` request from this endpoint will return a series of `FromRadio` protobufs.

The protobufs will be sent in binary as the body for the request.

Parameters

`/api/v1/fromradio?all`

`all=false` (unset default)

Only one protobuf is returned.

`all=true`

All available protobufs are returned.

`/api/v1/fromradio?chunked`

`chunked=false` (unset default, not yet implemented)

The request returns all protobufs that can be delivered for the client's session (this would allow the client to poll by doing a series of requests). This is the only option that is supported in the initial release.

`chunked=true` (not yet implemented)

If `chunked=true`, the response will be a stream of chunks that the server will keep open as long as the client wants. This will allow efficient streaming of new `FromRadio` protobufs as they are generated by the radio.

Authentication

There isn't **any** user authentication. We assume access to the HTTP server is enough to establish trust.

Client

JavaScript

See: <https://github.com/meshtastic/meshtastic.js>

A reference client written in JavaScript will provide a JavaScript API for using this transport. That client will do HTTP connections, use the generated protobuf JavaScript code and provide an API that hides all of this REST plumbing. The two key methods will be `sendToRadio(packet)` and `onFromRadio(callback)`.

Protoman

See: <https://github.com/spluxx/Protoman>

Protoman is able to interface with the Meshtastic REST API out of the box. This is useful for manual testing of the endpoints.

Security

HTTP and HTTPS are both supported on the ESP32 using self signed certificates on HTTPS.

Related documents

Interesting slide pack on the concept: <https://www.slideshare.net/mokeefe/javaone-2009-ts5276-restful-protocol-buffers>

Module API

The purpose of this tutorial is for writing new core modules that can be run on a device. In most cases, it is best to start with utilizing the serial module rather than creating a new one. However, if you're interested in creating a new core functionality from scratch, then building a module would be appropriate.

Key concepts

All modules should be sub-classes of `MeshModule`. By inheriting from this class and creating an instance of your new module - your module will be automatically registered to receive packets.

Messages are sent to particular port numbers (similar to UDP networking). Your new module should eventually pick its own port number (see below). For development use, you can simply use `PRIVATE_APP` (which is the default).

Packets can be sent and received as either:

Raw binary structures

Protobufs.

Class hierarchy

You will typically want to inherit from either SinglePortModule (if you are just sending/receiving raw bytes) or ProtobufModule (if you are sending/receiving protobufs). You'll implement your own handleReceived/handleReceivedProtobuf - probably based on the example code.

The relevant bits of the class hierarchy are as follows:

First Level: MeshModule

src/mesh/MeshModule.h - you probably don't want to use this base-class directly.

Second Level: SinglePortModule

src/mesh/SinglePortModule.h - for modules that send/receive from a single port number (the normal case).

Third Level: ProtobufModule

src/mesh/ProtobufModule.h - for modules that send/receive a single particular Protobuf type. Inherit from this if you are using protocol buffers in your module.

Startup Operations

If your module needs to perform any operations at startup you can override and implement the `setup()` method to run your code.

If you need to send a packet you can call `service.sendToMesh` with code like this (from the examples):

```
MeshPacket *p = allocReply();
p->to = dest;

service.sendToMesh(p);
```

Example Modules

A number of key services are implemented using the Module API, These modules are as follows:

TextMessageModule - Receives text messages and displays them on the LCD screen/stores them in the local DB.

NodeInfoModule - Receives/sends User information to other nodes so that usernames are available in the databases.

RemoteHardwareModule - A module that provides easy remote access to device hardware (for things like turning GPIOs on or off).

Intended to be a more extensive example and provide a useful feature of its own. See [remote-hardware](#) for details.

ReplyModule - A simple module that just replies to any packet it receives (provides a 'ping' service).

Getting started

The easiest way to get started is:

Build the firmware codebase.

Copy the ReplyModule as a template into `src/modules/`.

```
cp src/modules/ReplyModule.* src/modules/YourModule.*
```

Change the port number from `PortNum_REPLY_APP` to

`PortNum_PRIVATE_APP`.

Edit the `setupModules()` function located at `modules/Modules.cpp` to add a call to create an instance of your module (see comment at head of that function).

Rebuild with your new module and install on the device.

Use the Meshtastic Python CLI tool to send a packet to your board, for example:

```
meshtastic --dest 1234 --sendping
```

 where 1234 is another mesh node to send the ping to.

Threading

It is very common that you would like your module to be invoked periodically. We use a crude/basic cooperative threading system to allow this on any of our supported platforms. Simply inherit from `OSThread` and implement `runOnce()`. See the `OSThread` documentation for more details.

Sending messages

If you would like to proactively send messages (rather than just responding to them), just call `service.sendToMesh()`. For an example of this, see `NodeInfoModule::sendOurNodeInfo(...)`.

Picking a port number

See Meshtastic Port Numbers

How to add custom protocol buffers

If you would like to use protocol buffers to define the structures you send over the mesh (recommended), here's how to do that.

Create a new `.proto` file in the `protos` directory.

Run `./bin/regen-protos.sh` to regenerate the C code for accessing the protocol buffers. If you don't have the required nanopb tool, follow the instructions printed by the script to get it. Done! You can now use your new protobuf just like any of the existing protobufs in Meshtastic.

Critical Error Codes

The device might report these fault codes on the screen, but it will also be outputted on the device serial output. If you encounter a fault code, please post on the forum and we'll try to help.

 **NOTE**

This table is derived from the protobufs

Name	Number	Description
TxWatchdog	1	A software bug was detected while trying to send LoRa
SleepEnterWait	2	A software bug was detected on entry to sleep
NoRadio	3	No LoRa radio hardware could be found
Unspecified	4	Not normally used
UBloxInitFailed	5	We failed while configuring a UBlox GPS
NoAXP192	6	This board was expected to

Name	Number	Description
		have a power management chip and it is missing or broken
InvalidRadioSetting	7	The channel tried to set a radio setting which is not supported by this chipset, radio comms settings are now undefined
TransmitFailed	8	Radio transmit hardware failure. We sent data to the radio chip, but it did not reply with an interrupt
Brownout	9	We detected that the main CPU voltage dropped below the minimum acceptable value
SX1262Failure	10	Selftest of SX1262 radio chip failed
RadioSpiBug	11	A (likely software but possibly hardware) failure was detected while trying to send packets. If this occurs on your board, please post in the forum so

Name	Number	Description
		that we can ask you to collect some information to allow fixing this bug

Firmware



Building Firmware

Meshtastic uses PlatformIO, a development environment that enables easy multi-...



OLED Localization

1. Create an extended ASCII custom font. Use a glyph editor to create a new font ...



Port Numbers

Any new app that runs on the device or via sister apps on phones/PCs should pick...



Stacktraces

Decoding Stacktraces

Building Meshtastic Firmware

Meshtastic uses PlatformIO, a development environment that enables easy multi-platform development and centralized tooling.

Setup the Build Environment

Install PlatformIO

Clone the Meshtastic Firmware repository

```
git clone https://github.com/meshtastic/firmware.git
```

Update the repository's submodules

```
cd firmware && git submodule update --init
```

ⓘ INFO

If you want to build the RP2040 targets and get a 'Filename too long' error on Windows, please refer to the Platformio documentation for this toolchain

Build

Open the newly cloned folder in Visual Studio Code. If you do this

for the first time, this can take quite some while as PlatformIO will download all the necessary tooling and libraries. Also if platformio is not installed, VSCode will ask you to install it, probably requiring a restart of the program.

To select the device you wish to build, open your command palette:

Windows: `Ctrl + Shift + P`

Mac: `command + Shift + P`

Enter: `PlatformIO: Pick Project Environment` and select your target.

To build the firmware, simply run `PlatformIO: Build` from your command palette.

Finally, flash the firmware to your device by running `PlatformIO: Upload`

Adding Custom Hardware

The build system is modular. Adding a new board variant for an already supported architecture is straightforward.

Build with Custom Hardware

Go to the `variants` folder in the firmware source code and make a new directory for your hardware, let's call it `m5stack_atom` and copy an existing configuration you wanna modify:

```
cd variants; mkdir m5stack_atom  
cp heltec_v1/* m5stack_atom  
cd m5stack_atom
```

Modify the `platformio.ini` in this subdirectory from the canonical define of the hardware variant (`HELTEC_V1` in this case) to `PRIVATE_HW` and make the `-I` on the `build_flags` point to the newly created dir.

```
[env:m5stack-atom]  
extends = esp32_base  
board = m5stack-atom  
monitor_filters = esp32_exception_decoder  
build_flags =  
    ${esp32_base.build_flags} -D PRIVATE_HW -I  
variants/m5stack_atom  
lib_deps =  
    ${esp32_base.lib_deps}
```

Edit the `variant.h` file in this subdirectory to reflect the defines and configurations for your board. The example is very well commented.

Build, run and debug until you are satisfied with the result.

Distribute / Publish Custom Builds

Perform all of the steps building with custom hardware until your hardware runs fine.

Send a proposal to add a new board.

If approved, go to (<https://github.com/meshtastic/protobufs>) and send a Pull Request for the `mesh.proto` file, adding your board to the `HardwareModel` Enum.

Change your define in `platformio.ini` from `PRIVATE_HW` to `YOUR_BOARD`. Adjust any macro guards in the code you need to support your board.

Add your board identifier to `architecture.h` on the firmware repo in the folder of the platform you are using, and send in that Pull Request too.

Wait for the Pulls to be merged back into Master.

Profit :-)

OLED Localization Guide

Create an extended ASCII custom font. Use a glyph editor to create a new font file. The easiest way is to use the online glyph editor from the OLED library. (glyph editor source code)

- i. Copy and paste the existing font.
- ii. Modify it according desired codepage and save the new font file in `graphics/font` folder. Please note that the used font file format differs from common Adafruit GFX.

Update the `customFontTableLookup` function in `Screen.h`

- i. To map the double-byte UTF-8 code to the corresponding extended ASCII character of the desired codepage update the `customFontTableLookup` function in the `Screen.h` file.
- ii. Modify the `switch (last)` statement: use left byte from UTF-8 code in the `case` label to map character's right byte to its extended ASCII code by specifying an offset.

Define language and font in `Screen.cpp`

```
#ifdef OLED_{LANG_NAME}  
#include "fonts/OLEDDisplayFonts{LANG_NAME}.h"  
#endif  
  
...
```

Define language in `variant/*/platformio.ini`

```
build_flags =
${esp32_base.build_flags}
-D xxxxx
-D OLED_{LANG_NAME}
-I variants/xxxxx
```

Meshtastic Port Numbers

Any new app that runs on the device or via sister apps on phones/PCs should pick and use a unique "portnum" for their applications use.

The current list of port numbers can be found listed in the protobufs

Assignment

PortNums should be assigned by the following ranges:

Portnum	Usage
0-63	Core Meshtastic use, do not use for third party apps.
64-127	Registered 3rd party apps, send in a pull request that adds a new entry to portnums.proto to register your application
256-511	Use one of these portnums for your private applications that you do not want to register publicly

All other values are reserved.

Integration

If you are making a new app using Meshtastic, please send a pull request to add your chosen "portnum" to this master table.

Stacktraces

Decoding Stacktraces

You may encounter a situation where your device crashes and are left with a stacktrace, below are two methods of decoding them.

Manually



This method uses the symbols of the `firmware.elf` file generated from your latest build, you may wish to rebuild to get up-to-date symbols.

Save the backtrace string to a text file:

```
backtrace.txt
```

```
Backtrace: 0x....
```

Run the exception decoder:

```
bin/exception_decoder.py backtrace.txt
```

In Real-Time

In order to decode stack traces in real time, keep the following

command (replacing `DEVICE_PORT` with your device's port)
running in your terminal with the target device connected:

```
pio device monitor --port DEVICE_PORT -f  
esp32_exception_decoder
```

Web Client Development

Overview

The Meshtastic web interface can be hosted or served from a node.

The official hosted version can be found at
<https://client.meshtastic.org>.

The version served from a node can be accessed by first connecting your node to your network and then navigating to `http://meshtastic.local` (or `your_node_ip.local`).

Development & Building

Development

Clone the Meshtastic Web Repo repository

```
git clone https://github.com/meshtastic/web.git
cd web
```

Install the dependencies.

```
pnpm i
```

Building

Build the project:

```
pnpm build
```

Start the development server:

```
pnpm dev
```

Packaging

Build the project:

```
pnpm build
```

GZip the output:

```
pnpm package
```

Releases

Releases are automatically generated for every commit as per out

CI. This performs two actions:

Generates a perpetually updated GitHub release with an accompanying `build.tar` that is automatically get's pulled by the firmware CI at build time.

A hosted version is deployed to client.meshtastic.org.

Meshtastic Python Development

A note to developers of this lib

We use the Visual Studio Code (VScode) default python formatting conventions (autopep8). So if you use that IDE you should be able to use "Format Document" and not generate unrelated diffs. If you use some other editor, please do not change formatting on lines you have not changed yourself.

Building

To build a new release

```
apt install pandoc  
sudo pip3 install markdown pdoc3 webencodings  
pyparsing twine autopep8 pylint pytest pytest-cov
```

For development

```
pip3 install -r requirements.txt
```

Linting

```
pylint meshtastic
```

Testing

Install and run pytest

For more verbosity, add `-v` or even `-vv`

```
pip3 install .
pytest -vv
```

Run just unit tests

```
pytest
# or (more verbosely)
pytest -m unit
# or
make
```

Run just integration tests

```
pytest -m int
```

Run the smoke test with only one device connected serially (aka smoke1)

```
pytest -m smoke1
```

⚠ CAUTION

Running `smoke1` will reset values on the device, including the region to 1 (US). Be sure to hit the reset button on the device after the test is completed.

Run the smoke test with only two device connected serially (aka smoke2)

```
pytest -m smoke2
```

Run the wifi smoke test

```
pytest -m smokewifi
```

Run a specific test

```
pytest -msmoke1 meshtastic/tests/  
test_smoke1.py::test_smoke1_info
```

or to run a specific smoke2 test

Add another classification of tests such as `unit` or `smoke1`

See `pytest.ini`.

To see the unit test code coverage

```
pytest --cov=meshtastic
# or if want html coverage report
pytest --cov-report html --cov=meshtastic
# or
make cov
```

To see slowest unit tests, you can run

```
pytest --durations=0
# or
make slow
```

Generate the Python API documentation

Pre-generated: API documentation

```
bin/regen-docs.sh
```

Wire encoding

When sending protobuf packets over serial or TCP each packet is

preceded by uint32 sent in network byte order (big endian). The upper 16 bits must be 0x94C3. The lower 16 bits are packet length (this encoding gives room to eventually allow quite large packets).

Implementations validate length against the maximum possible size of a BLE packet (our lowest common denominator) of 512 bytes. If the length provided is larger than that we assume the packet is corrupted and begin again looking for 0x4403 framing.

The packets flowing towards the device are ToRadio protobufs, the packets flowing from the device are FromRadio protobufs. The 0x94C3 marker can be used as framing to (eventually) resync if packets are corrupted over the wire.

Note: the 0x94C3 framing was chosen to prevent confusion with the 7 bit ascii character set. It also doesn't collide with any valid utf8 encoding. This makes it a bit easier to start a device outputting regular debug output on its serial port and then only after it has received a valid packet from the PC, turn off unencoded debug printing and switch to this packet encoding.

Building

A python release consists of publishing the release to PyPi
<https://pypi.org/project/meshtastic/> as well as producing single-executable files that are downloadable from Github
<https://github.com/meshtastic/Meshtastic-python/releases>.

Pre-requisites

No pre-requisites are needed locally to make a release. All builds are done via Github Actions currently.

To test/validate, you will need to run:

```
pip3 install -r requirements.txt
pip install .
```

Instructions

Update protobufs by running the "Update protobufs" workflow in Actions: https://github.com/meshtastic/Meshtastic-python/actions/workflows/update_protobufs.yml

run the "smoke1" test (optional):

connect one device to the serial port and run:

```
pytest -m smoke1
```

run unit tests: `pytest` (optional)

run `bin/test-release.sh` (optional)

Run the "Make Release" workflow in Actions: <https://github.com/meshtastic/Meshtastic-python/actions/workflows/release.yml>

After the "Make Release" is done, go into Releases:

<https://github.com/meshtastic/Meshtastic-python/releases> There should be a draft. Add the title, update the "What's Changed" (Tip: Click on the "Auto-generate release notes" button.). Uncheck the "This is a pre-release" (if applicable).

 **NOTE**

You need permissions in the GitHub project to make a build

Instructions - automated

Go to Actions / Make Release / Run Workflow <https://github.com/meshtastic/Meshtastic-gui-installer/actions/workflows/release.yml>

Draft & Publish release <https://github.com/meshtastic/Meshtastic-gui-installer/releases>

Using the Meshtastic Python Library

An example using Python 3 code to send a message to the mesh, get and set a radio configuration preference:

```
import meshtastic
import meshtastic.serial_interface

# By default will try to find a meshtastic device,
# otherwise provide a device path like /dev/ttyUSB0
interface = meshtastic.serial_interface.SerialInterface()
# or something like this
# interface =
meshtastic.serial_interface.SerialInterface(devPath='/dev/
cu.usbmodem53230050571')

# or sendData to send binary data, see documentations for
other options.
interface.sendText("hello mesh")

ourNode = interface.getNode('^local')
print(f'Our node preferences:{ourNode.localConfig}')

# update a value
print('Changing a preference...')
ourNode.localConfig.position.gps_update_interval = 60
```

Another example using Python 3 code to send a message to the mesh when WiFi is enabled:

```
import time
import meshtastic
import meshtastic.tcp_interface
from pubsub import pub

def onReceive(packet, interface): # called when a packet arrives
    print(f"Received: {packet}")

def onConnection(interface, topic=pub.AUTO_TOPIC): # called when we (re)connect to the radio
    # defaults to broadcast, specify a destination ID if you wish
    interface.sendText("hello mesh")

pub.subscribe(onReceive, "meshtastic.receive")
pub.subscribe(onConnection,
"meshtastic.connection.established")
interface =
meshtastic.tcp_interface.TCPIInterface(hostname='192.168.68.74')
while True:
    time.sleep(1000)
interface.close()
```

Note: Be sure to change the IP address in the code above to a valid IP address for your setup.

For the rough notes/implementation plan see TODO. See the API for full details of how to use the library.

A note to developers of this lib

We use the visual-studio-code default python formatting conventions (autopep8). So if you use that IDE you should be able to use "Format Document" and not generate unrelated diffs. If you use some other editor, please don't change formatting on lines you haven't changed.

If you need to build a new release you'll need:

Command

```
apt install pandoc  
sudo pip3 install markdown pandoc webencodings  
pyparsing twine autopep8
```

Getting Started with Meshtastic.js



TIP
Full API documentation is available at js.meshtastic.org

Intro

Meshtastic.js is a JavaScript library that provides an interface to Meshtastic devices. It can be used to build applications to interface with a Meshtastic network. Currently, HTTP(S) and Bluetooth connections are supported.

If you wish to view the code or contribute to development of the library, please visit the JavaScript code GitHub page.

Connection methods

HTTP(S)

Bluetooth

Serial

Connecting to a device

```
import type React from "React";

import { IHTTPConnection } from "@meshtastic/js";

export const Connection = (): JSX.Element => {
  const connection = new IHTTPConnection();

  const connect = (): void => {
    void connection.connect({
      address: "10.0.0.10",
      fetchInterval: 3000
    });
  };

  return <button onClick={connect}>Connect
Bluetooth</button>;
};
```

Subscribing to device events

Preface

The event system that the library uses is provided by RxJS, this is the fundamental way in which to interact with devices and the mesh network.



TIP
Full guide to using RxJS is available at rxjs.dev

...



INFO
Many of the packet types that the device is designed to send are automatically processed and decoded for you in their own event stream

Events

Heartbeat

Device Status

Device Translation

From Radio

Data Packet

My Node Info

Radio Config

Node Packet

Position Packet

Text Packet

HTTP API

ⓘ INFO

Methods and types for using the device HTTP API are exported

```
import {
  Client,
  Types,
  Protobuf,
  SettingsManager
} from "@meshtastic/js";

/**
 * Connection method
 */

const client = new Client();
SettingsManager.setDebugMode(Protobuf.LogLevelEnum.DEBUG);

const connection = client.createHTTPConnection();
connection.connect("192.168.x.x");

const restartDevice: Promise<void> =
  connection.restartDevice();
const getStatistics: Promise<void | Types.WebSPIFFSResponse> =
  connection.getStatistics();
const getNetworks: Promise<void | Types.WebNetworkResponse> =
```


Maintaining Documentation

Meshtastic documentation is an important ingredient to the overall project. We want users to hit the ground running with the information they need right at their finger tips. This section will discuss the documentation software stack, file organization, and style guides.

Software Stack

All of our documentation resides on GitHub. Instructions for setting up your GitHub account are located [here](#).

Our documentation is powered by Docusaurus — a documentation platform built on React that utilizes markdown files. Because markdown files are easy to edit, most content changes should be fairly simple.

Another component that we use is Vercel — a platform for front-end frameworks and static sites. Instructions for setting up your instance of Vercel are located [here](#).

Documentation Organization

Section	File Path	Description
About Meshtastic	docs/ about	A high level explanation of Meshtastic plus everything relating to the Meshtastic mesh. This includes radio settings, mesh algorithm and encryption.
Getting Started	docs/ getting- started	Instructions on how to get the Meshtastic firmware onto a users device.
Device Settings	docs/ settings	Details for both the device and module configurations. Details each user setting and offers explanations on their functionalities in addition to guiding the user on how to configure the device using the various clients available (Android, CLI, iOS, Web).
Hardware Details	docs/ hardware	Any hardware related content such as officially supported radios and

Section	File Path	Description
		their peripherals such as 3d printed cases, antennas, buttons, chime, rotary encoders, and screens.
Meshtastic Software	docs/ software	An overview of the current software used in conjunction with Meshtastic to include officially supported client and community applications.
Contribute to Meshtastic	docs/ developers	Details of the necessary information needed for developers to start contributing to the development of the Meshtastic project.
Legal	docs/ legal	Any legal information. Most changes here will be handled by developers actually working on the projects that require any legal disclosures. Examples include: the Meshtastic trademark, terms of service, and privacy policy.

Quick Start

Assuming you have the prerequisites installed, running a local instance of Docusaurus takes three steps:

Fork/Clone the meshtastic/meshtastic repository and navigate to the root directory of the project.

Clone the project

```
git clone https://github.com/meshtastic/  
meshtastic.git
```

Clone fork of the project

```
git clone  
https://github.com/[username]/meshtastic.git
```

Change Directory

```
cd ~/Meshtastic
```

Install Dependencies

Install dependencies using Yarn

```
yarn install
```

Run Docusaurus

```
Run node.js server
```

```
yarn start
```



TIP

Before submitting a pull request, it's helpful to run the following command to ensure there are no broken links or errors:

```
Build Project
```

```
yarn build
```

Style Guides



Markdown Features

Overview



Config Pages

Overview

Markdown Features

Overview

We have developed several React components for assisting with writing documentation.

Features

Light/Dark Mode Switch

Usage

```
import { Dark, Light } from '/src/components/ColorMode';
<Dark>
  <p>Dark</p>
</Dark>
<Light>
  <p>Light</p>
</Light>
```

Demo

This is only shown in light mode.



Code Blocks

Usage

Always specify the language used directly after the start of the code block (```).

NOTE

For command line examples, please use `shell` and not any of the other aliases.

for further information please see the relevant Docusaurus page

```
```ts title="Demo"
export const typedArrayToBuffer = (array:
Uint8Array): ArrayBuffer => {
 return array.buffer.slice(
 array.byteOffset,
 array.byteLength + array.byteOffset
);
```

## Demo

### Demo

```
export const typedArrayToBuffer = (array:
Uint8Array): ArrayBuffer => {
 return array.buffer.slice(
 array.byteOffset,
 array.byteLength + array.byteOffset
);
};
```

# Config Pages

## Overview

Setting pages should focus solely on settings and configuring the device. Hardware is often a related topic, however these pages should not attempt to explain attaching hardware. Mention that hardware is required and link to the appropriate page(s).

## Layout

### Overview

This section should describe the group of settings and what they do for the device.

### Settings

This section starts with an alphabetized table of settings, available values, and default values.

Available values should be listed with tick marks `` surrounding the value.

Value Type	Example
Boolean	true, false
List	apple, banana, orange
Range	0 - 100

After the table each setting is described in brief detail. If the available options for a setting need additional explanation, a table should be used to describe each available option.

## Details

If additional details are needed, this optional section can explain that. These would include prerequisites, links to hardware guides, etc.

## Examples

This optional section can have examples of configurations needed where multiple settings are required to be set up for a specific use case.

# Example Template

Template for Settings Pages

```

```

```
id: unique-id
title: Title for Page
sidebar_label: Label for Sidebar

```

```
<!---- Allows client-specific tabs to be used ---->
```

```
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';
```

```
Overview
```

```
<!---- Add overview text to describe this group of settings ---->
```

```
Settings
```

```
<!---- Table of settings in alphabetical order ---->
```

Setting	Acceptable Values
Default	
my_setting_with_options	`apple`, `banana`,





# Serving Docs Locally for Development

 **NOTE**

Some things won't display properly like logos or protobufs page, this is not cause for concern.

## Prerequisites

In order to set up your local environment, you will need to install:

Node.js Runtime

PNPM Package Manager

## Getting Started

### Fork the Meshtastic Repository

Log into GitHub and create a fork of the meshtastic/meshtastic repository.

## Clone your Meshtastic Repository fork

 **NOTE**

Replace `username` with your GitHub username.

Clone `username/Meshtastic Repo`

```
git clone https://github.com/username/meshtastic.git
```

## Change directory to Local copy

Change Directory

```
cd ~/meshtastic
```

## Install Dependencies

Install dependencies using pnpm

```
pnpm i
```

## Run Development Server

Run node.js server

```
pnpm start
```

 **TIP**

Before submitting a pull request, it's helpful to run the following command to ensure there are no broken links or errors:

Build Project

```
pnpm build
```

## Update Local Repository

### Verify Upstream Remote is Set

Check Remote and Upstream Repositories

```
git remote -v
```

If it's set, skip to Align with meshtastic/meshtastic Master branch

### Update/Set Upstream if it isn't configured properly

If upstream exists, set the URL:

## Update Upstream Repository

```
git remote set-url upstream https://github.com/
meshtastic/meshtastic.git
```

If upstream doesn't exist, add the URL:

## Add Upstream Repository

```
git remote add upstream https://github.com/
meshtastic/meshtastic.git
```

## Align with **meshtastic/meshtastic** Master branch

### CAUTION

This will delete any unfinished work. Make sure that you've saved and committed any work you wish to push up to your fork.

### INFO

The following command assumes the clone of your Meshtastic fork is in the home directory (`~/meshtastic`). Adjust the path to the correct path on your machine.

Rebase local Meshtastic to remote Meshtastic

```
cd ~/meshtastic ; git fetch upstream ; git checkout master ; git rebase upstream/master
```

# Publishing Meshtastic.org

## Publish Live

Generate protobuf docs

```
cd meshtastic
./scripts/gen-proto-docs.sh
```

Build

```
pnpm build
```

Submit Pull Request

## Publish to Vercel

Setting up a Vercel account is an optional step that can help you view your changes to the documentation in a server environment just like the actual docs. This can be helpful to include when submitting a pull request so that developers can review your changes and visually check that things look correct.

## Set up Vercel account

Go to [vercel.com](https://vercel.com)

Login with your GitHub account, click [Continue with GitHub](#)

## Link your fork of the project

Click [New Project](#)

Under [Import Git Repository](#) select [+ Add GitHub Account](#)

You'll be redirected to GitHub to allow access to select repositories.

Select your fork of the project: [username/Meshtastic](#)

## Configure project

Configure project:

Set a name for the project

Select a framework preset: [Docusaurus 2](#)

Click [Deploy](#)

That's it! You should now see your project with a green or orange status dot showing that your fork of the project has been compiled. There will be a commit-specific url that you can share to view your changes. There also will be a branch-specific url that you can view. If there are any errors it will show up red and include the logs for you to figure out what has gone wrong.

 **TIP**

There is a limited number of branch urls that you will be able to view. If you notice that option has disappeared, you can delete unused branches on your fork and that will enable that feature again.

Branch urls are helpful in PRs because they will remain constant, and you won't need to resubmit a new url for review each new commit if changes are requested.

# Reference Material



## Protobufs

Overview



## LoRa Datasheet

Useful Resources



## GitHub

Overview



## GNSS Modules

A detailed contribution from community contributor GPSFan which provides an in...

# Protobufs

## Overview

Protocol Buffers, commonly referred to as Protobufs, are a language-neutral, platform-neutral, extensible mechanism for serializing structured data. They are used by Meshtastic software for encoding and transmitting data between the App and Device, as well as for Device-to-Device communication.

Protobufs provide an efficient and lightweight way of exchanging data, making them well-suited for use in resource-constrained environments like the Meshtastic network. They offer several advantages over traditional data formats like XML or JSON, including:

**Smaller serialized size:** Protobuf serialized data is typically much smaller than XML or JSON representations of the same data.

**Faster serialization and deserialization:** Protobufs are designed to be serialized and deserialized quickly, which is important for applications that need to process large amounts of data.

**Type-safe and self-describing:** Protobuf messages are type-safe, and the message formats are self-describing, making it easier to work with and maintain the data over time.

# **Meshtastic Protobufs**

The Meshtastic project defines its own set of Protobuf messages for various types of data exchanged between app-device and device-device. These messages are organized into different modules.

The official documentation for the Meshtastic Protobuf messages can be found on the Buf Schema Registry (BSR). The BSR provides a centralized location for managing and documenting the Protobuf schemas used by the project.

# LoRa Design Guide

## Useful Resources

### LoRa Modem Design Guide

A guide from Semtech explaining the key principles and design parameters behind LoRa modulation and their SX1272/3/6/7/8 LoRa modem chips. Helpful for understanding core LoRa concepts like spreading factor, bandwidth, time on air, sensitivity etc.

# GitHub

## Overview

The Meshtastic project is hosted on GitHub, a popular web-based platform for version control and collaborative software development. GitHub allows developers to manage and track changes to their code, collaborate with others, and distribute their work.

If you want to contribute to the Meshtastic project or simply stay up-to-date with the latest developments, you'll need a GitHub account.

## Setting up a GitHub Account

Go to [github.com](https://github.com)

Click on the "Sign up" button in the top right corner.

Follow the on-screen instructions to create your account.

With a GitHub account, you can:

Access and view the Meshtastic project repositories.

Report issues or bugs you encounter.

Propose changes or new features through pull requests.

Discuss and collaborate with other contributors.

# Meshtastic on GitHub

The Meshtastic project is organized under the meshtastic GitHub organization. Here are some of the main repositories:

firmware: The firmware code for Meshtastic devices.

Meshtastic-Android: The Android application for Meshtastic.

web: The Meshtastic Web Client.

protobufs: The Protocol Buffer definitions used by Meshtastic.

Meshtastic-Apple: Apple iOS, iPadOS & macOS Clients For Meshtastic.

meshtastic: The Meshtastic project website and documentation.

python: The Python CLI and API for communicating with Meshtastic devices.

## Contributing to Meshtastic

If you're interested in contributing to the Meshtastic project, you'll need to follow the contribution guidelines outlined in the project's repositories. These guidelines typically include instructions for setting up your development environment, coding conventions, and the process for submitting pull requests.

Additionally, many Meshtastic repositories include a `README.md` file that provides an overview of the project, installation

instructions, and other relevant information.

# README Template

Meshtastic developers are encouraged to follow a consistent README format for new repositories. The template includes sections for an overview, getting started guide, documentation/API references, installation instructions, and compatibility information.

```
Repository Name

<!--Project specific badges here-->

<!--Crowdin Badge can be generated from https://crowdin.meshtastic.org/project/badge?repo=meshtastic-->

[![Crowdin](https://badges.crowdin.net/e/<badge_id>/localized.svg)](https://crowdin.meshtastic.org/project/badge?repo=meshtastic)

[![CI](https://img.shields.io/github/actions/workflow/status/meshtastic/<repo>/ci.yml?branch=master&label=actions&logo=github&style=flat)](https://github.com/meshtastic/meshtastic/actions)

[![CLA assistant](https://cla-assistant.io/readme-badge/meshtastic)](https://cla-assistant.io/meshtastic/meshtastic)

[![Fiscal Contributors](https://opencollective.com/meshtastic/tiers/badge.svg?label=Fiscal%20Contributors&color=deeppink)](https://opencollective.com/meshtastic/tiers)

[![Vercel](https://img.shields.io/static/v1?label=Powered%20by&message=Vercel&style=flat&logo=vercel&color=black)](https://vercel.com/meshtastic)

Overview
```



# Understanding GNSS Modules

A detailed contribution from community contributor GPSFan which provides an in-depth look at GNSS modules, specifically focusing on u-blox modules, which could be useful for hardware devs designing devices for integrating GPS functionality into Meshtastic hardware projects.

## u-blox Module Types

### Clones

Clones, like Beitian, Goouu Tech, BZGNSS & VKEL, can be functionally equivalent to u-blox parts, and will have their own label on the module.

Sometimes, clones have the proper amount of flash and can be updated as new firmware comes out, sometimes not.

### Counterfeits

Counterfeits usually use u-blox chips inside but have a u-blox looking label on the module with substandard circuitry inside.

## **Fakes**

Fakes have a u-blox looking label, but all bets are off as to what's inside, often another Chinese chip or a 6010 with an M8 label.

Beware, most modules seen on Amazon, eBay, Banggood, Temu or AliExpress are in one of the above three categories.

## **Genuine u-blox Modules**

Digikey, Mouser, Arrow, u-blox and others sell genuine u-blox parts at premium prices.

## **u-blox Chip Series**

### **Neo-6 Series**

The Neo-6 is the oldest (although there are older u-blox 4 and 5 parts), most power hungry, least capable and lowest sensitivity module. The 6010 chip supports 50 channels.

### **Neo-7 Series**

The Neo-7 supports SBAS, GLONASS as well as GPS, and has some nice high precision parts (Neo-7P). The 7020 chip supports 56 channels.

## **M8 Series**

The M8 supports GPS, SBAS, GLONASS, QZSS, BeiDou and Galileo, but can only support 3 major ones concurrently. QZSS and GPS should always be either enabled together or both disabled for M8 & above parts. The 8030 chip supports 72 channels.

## **M9 Series**

The M9 supports all the above constellations but can use 4 systems at once (SBAS and QZSS are augmentation systems and don't count in this number). The 9140 chip forms the basis of the M9, D9 and F9 products, the F9P being a very capable L1/L2 or L5 RTK capable product. The 9140 chip supports 92 channels.

## **M10 Series**

It is left as an exercise for the reader to read the product briefs for the M10 series. The 10050 chip supports 72 channels.

## **Comparison to Other GNSS Chips**

As a comparison:

The Unicore UM980 has 1408 channels

The Septentrio mosaic-T has 448 "hardware channels"

# u-blox Chip Configurations

The u-blox chips have several configurations that can be customized by the module maker:

- Flash size
- LNA
- TCXO
- SAW filter
- General circuit layout

# u-blox Firmware and Protocol

Each chip series supports a different and increasing protocol version. Beginning with 23.01, the legacy CFG commands were replaced with a different config method using the VALSET/VALGET/VALDEL series of commands. However, even up to protocol 34, many of the CFG commands still work. The new config method allows much finer grained configuration at the cost of complexity. Trying to support both legacy and new config methods can be challenging.

## Protocol Specification Levels

There are 3 levels of protocol specifications:

**Internal Use Only:** No one but u-blox employees see these, and

they detail the entire firmware command set.

**NDA Restricted:** OEMs that buy lots of parts and sign an NDA have access to these. Few if any make it out into the wild, and they detail a subset of the internal specs.

**Public:** Can be downloaded by anyone off the u-blox website. These are a subset of the NDA restricted, and often contain errors and omissions.

## Libraries and Hidden Commands

All u-blox firmwares support hidden or undocumented commands. SparkFun has a u-blox config library that uses the new method, it is very complete and very much overkill.

## Future u-blox Chips

There are new u-blox chips/modules in the pipeline, and in R&D, competing with the Chinese designed and produced parts like the UM980 and UC6580 should produce better and cheaper parts from u-blox (one would hope).

# Common GPS Problems

## Self-inflicted Problems

### Attempting Indoor Fix

Trying to get a fix indoors is never recommended.

The GNSS signals are very weak, and anything between the antenna and the satellite (even the atmosphere) will degrade the signal.

All indoor locations are not created equally in terms of signal reception.

### Unrealistic Fix Time Expectations

Expecting a fix immediately after power on is unrealistic.

The time to first fix (TTFF) varies depending on whether it's a cold start (receiver has no time/almanac/ephemeris data), warm start, or hot start.

A good receiver under ideal conditions can take up to 28 seconds for a cold start fix.

### Impatient Timeout

Waiting too long for a fix when receiver parameters may have timed out, meaning it will never get a fix after that timeout period.

## **Using Sub-optimal Constellations**

Using only one or two constellations when the receiver can receive many is a waste of hardware resources.

## **Poor Antenna Placement**

GNSS antennas are directional and don't have much gain.

Putting the receiver in your pocket or not pointing the antenna towards the sky will reduce effectiveness.

# **Other Causes**

## **Hardware Design and Build Quality**

GNSS receivers operate at microwave frequencies, so signal path impedance and noise management are important.

Using a good external antenna with LNA and SAW filter can reduce locally generated noise.

Proper coaxial cables are also crucial.

## **Cost Cutting**

An overly aggressive approach to cost reduction can degrade performance incrementally to the point of making the receiver useless.

## **Aggressive Power Management**

Most receivers have aggressive power management modes that

can hamper acquisition/tracking if combined with a poor view of the sky.

### **Incorrect Initialization**

Supporting multi-generational products and different receiver manufacturers makes properly initializing the receiver challenging.

# Legal

## Disclaimers

**Project Status:** Meshtastic is now at a stage where it offers a *mostly* stable and reliable functionality, thanks to the dedicated efforts of our development team. While the core aspects of the project are well-established, we are still rapidly evolving and actively adding new features. As we continue to innovate and expand, some features in our current builds are in development, reflecting our commitment to continuous improvement and growth. We appreciate your engagement as we advance further in this exciting journey.

**Safety and Regulations:** Please be aware that the devices related to this project are not manufactured by us and have not been tested by regulatory organizations such as UL or the FCC. Therefore, using these devices should be considered experimental. While we focus on safety in our designs, we advise users to exercise caution to avoid any potential risks.

**Encryption and Security:** Our encryption measures are robust and a significant aspect of our project. However, it's important to recognize the limitations and risks associated with any encryption technology. For further understanding of these aspects, kindly review encryption.

# Legal FAQ

## **Q: Do you have plans to commercialize this project**

**A:** We do not have plans to commercialize Meshtastic. However, we are open to and supportive of others using our software in their commercial endeavors, provided they comply with the GPL license. More details can be found on our licensing guidelines.

## **Q: Does this project use patented algorithms?**

(Kindly borrowed from the geeks at ffmpeg)

We do not know, we are not lawyers so we are not qualified to answer this. Also, we have never read patents to implement any part of this, so even if we were qualified we could not answer it as we do not know what is patented. Furthermore, the sheer number of software patents makes it impossible to read them all, so no one (lawyer or not) could answer such a question with a definite no. We are merely geeks experimenting on a fun and free project.

# **Contributor Covenant Code of Conduct**

## **Our Pledge**

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

## **Our Standards**

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our

mistakes, and learning from the experience  
Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

The use of sexualized language or imagery, and sexual attention or advances of any kind

Trolling, insulting or derogatory comments, and personal or political attacks

Public or private harassment

Publishing others' private information, such as a physical or email address, without their explicit permission

Other conduct which could reasonably be considered inappropriate in a professional setting

## **Enforcement Responsibilities**

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when

appropriate.

## Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official email address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

## Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement using one or more of the methods:

In Discord, send a private message to an Admin

In Email, send a message to all of these:

[conduct@meshtastic.org](mailto:conduct@meshtastic.org)

[jm@meshtastic.org](mailto:jm@meshtastic.org)

[sachaw@meshtastic.org](mailto:sachaw@meshtastic.org)

All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and

security of the reporter of any incident.

# Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

## 1. Correction

**Community Impact:** Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

**Consequence:** A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

## 2. Warning

**Community Impact:** A violation through a single incident or series of actions.

**Consequence:** A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media.

Violating these terms may lead to a temporary or permanent ban.

### **3. Temporary Ban**

**Community Impact:** A serious violation of community standards, including sustained inappropriate behavior.

**Consequence:** A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

### **4. Permanent Ban**

**Community Impact:** Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

**Consequence:** A permanent ban from any sort of public interaction within the community.

## **Attribution**

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at <https://www.contributor-covenant.org/>

[version/2/1/code\\_of\\_conduct.html](#).

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at <https://www.contributor-covenant.org/faq>. Translations are available at <https://www.contributor-covenant.org/translations>.

# Meshtastic privacy policy

We don't collect any personal identifying information. We never capture usernames, the contents of your texts, or your location data.

If you opt-in to analytics on the Android app (thank you - that helps us know what things we need to improve), we will receive anonymized information about user behavior. This includes crash reports, screens used in the app, etc... Analytics is provided by Firebase Crashlytics.

Maps provided by Mapbox require analytics to work. For more information about what they collect, please see the [Mapbox privacy policy](#).

The search engine for this website is provided by Algolia, please see their [privacy policy](#) for details of what information they collect.

This is an open-source project run by hobbyists, and we try to be completely transparent. If you have questions on this policy, please post on the forum, and we'll reply/clarify/correct.

Keep being awesome!

# Licensing & Trademark Rules

Meshtastic® is a registered trademark of Meshtastic LLC.

Meshtastic software components are released under various licenses, see GitHub for details. No warranty is provided - use at your own risk.

## Licensing

You are hereby granted a two-year, non-exclusive license to use the Meshtastic® logo and trademark on your products or projects, in accordance with our trademark guidelines. While there is no fee for this usage, we suggest allocating \$1 per unit sold to a fund that will eventually support Meshtastic. This contribution, though not mandatory, is greatly appreciated.

Please ensure the inclusion of the following statement in your support documentation:

Meshtastic® is a registered trademark of Meshtastic LLC.

Meshtastic software components are released under various licenses, see GitHub for details. No warranty is provided - use at your own risk.

Alternatively, if you prefer, you can use our community project

logos, "M-Powered" and "M-PWRD", which are specifically designed for projects that utilize Meshtastic technology. These logos offer a straightforward way to show compatibility with our platform without needing a trademark grant. For more information on these logos, including usage guidelines and design files, please refer to Community Project Logos: M-Powered & M-PWRD.

## **Meshtastic Trademark Policy & Brand Guidelines**

Meshtastic, and Meshtastic logo ("Meshtastic Logo"), either separately or in combination, are hereinafter referred to as "Meshtastic Trademarks" and are trademarks of the Meshtastic LLC. Except as provided in these guidelines, you may not use the Meshtastic Trademarks or any confusingly similar mark as a trademark for your product, or use the Meshtastic Trademarks in any other manner that might cause confusion in the marketplace, including but not limited to in advertising, on websites, or on software. In fact, the law obligates trademark owners to police their marks and prevent the use of confusingly similar names by third parties. If you have questions about this policy, please contact the Trademark Supervisor by enquiring at [trademark@meshtastic.org](mailto:trademark@meshtastic.org).

# **Usage That Does Not Require Written Permission**

Below are the guidelines for use of the Meshtastic Trademarks where, as long as you are in compliance with the guidelines, no advance written permission is necessary. In all cases, use is permitted only provided that:

the use is not disparaging to Meshtastic or software distributed by Meshtastic.

the use does not imply sponsorship or endorsement by Meshtastic. proper trademark symbols are used in connection with the Meshtastic Trademarks and the trademark attribution statement must appear as explained in Proper Trademark Use.

the Logo Usage Guidelines are strictly observed.

## **Noncommercial and community web sites**

In the past, community members have inquired whether it is permissible to show support for Meshtastic by:

placing the Meshtastic Trademarks on a personal web site or blog to support open source software.

making a page on a social networking web service to support open source software.

linking to the Meshtastic website to provide information or show

support for open source software.

The guidelines relating to such usage are set forth in this section.

It is permissible to use the Meshtastic Trademarks on websites to show your support for the open source software, provided that:

where possible, the design logo hyperlinks to the Meshtastic website, <http://meshtastic.org/>, or if that is not possible, the site includes a prominent link to the Meshtastic website at <http://meshtastic.org/>.

proper trademark symbols are used in connection with the Meshtastic Trademarks and the trademark attribution statement must appear as explained in Proper Trademark Use.

the Logo Usage Guidelines are strictly observed.

the site indicates clearly that it is not affiliated with or endorsed by Meshtastic; in addition, where possible: the site must include the text "This site is not affiliated with or endorsed by the Meshtastic project" prominently on any page that includes the Meshtastic Trademarks, and

if the Meshtastic Trademarks appear in a page header or any area that is designed to be presented on more than one page, the notice must also be designed to be presented on all of those pages as well. (i.e., if the Meshtastic Trademarks appear in a site-wide header, the informational text must appear in that header or an identically site-wide footer.)

the site does not use visual styling that could be confusing to

viewers or visitors as to whether the site is hosted by or on behalf of Meshtastic.

A copy of the use of the Meshtastic Trademarks is provided to the Trademark Supervisor within seven (7) days of its initial use, for example by sending a URL or other copy of such use.

## **Business web sites**

In the past, community members have inquired whether it is permissible to show support for Meshtastic by:

displaying a link to the Meshtastic website using the Meshtastic Trademarks from a business web site.

displaying the Meshtastic Trademarks as part of a business that utilizes Meshtastic software.

The guidelines relating to such usage are set forth in this section.

It is permissible to use the Meshtastic Trademarks on business web sites, provided that:

the web site has non-Meshtastic primary branding.

the design logo hyperlinks to the Meshtastic website,  
<http://meshtastic.org/>.

the use does not imply sponsorship or endorsement by Meshtastic.

the use of the Meshtastic Trademarks does not imply an association with nor any form of endorsement by Meshtastic.

proper trademark symbols are used in connection with the

Meshtastic Trademarks and the trademark attribution statement must appear as explained in Proper Trademark Use.

the Logo Usage Guidelines are strictly observed.

the site does not use visual styling that could be confusing to viewers or visitors as to whether the site is hosted by or on behalf of Meshtastic.

a copy of the use of the Meshtastic Trademarks is provided to the Trademark Supervisor within seven (7) days of its initial use, for example by sending a URL or other copy of such use.

## Promotional events

In the past, community members have inquired whether it is permissible to use the Meshtastic Trademarks to promote events. The guidelines relating to such usage are set forth in this section.

It is permissible to use the Meshtastic Trademarks in such promotional events, provided that:

the use does not imply sponsorship or endorsement by Meshtastic.

the use of the Meshtastic Trademarks does not imply an association with or endorsement of the event or the goods distributed at such event.

proper trademark symbols are used in connection with the Meshtastic Trademarks and the trademark attribution statement must appear as explained in Proper Trademark Use.

the Logo Usage Guidelines are strictly observed.

A copy of the use of the Meshtastic Trademarks is provided to the Trademark Supervisor within seven (7) days of its initial use, for example by sending a URL or other copy of such use.

If you would like to make some non-software goods to give away or sell at the event and don't already have a license to do so, see Non-software goods.

## **Publications**

It is permissible to use the Meshtastic Trademarks in the title and content of a publication, provided that:

the use is clearly in reference to Meshtastic.

the use does not imply sponsorship or endorsement of the publication by Meshtastic.

proper trademark symbols are used in connection with the Meshtastic Trademarks and the trademark attribution statement must appear as explained in Proper Trademark Use.

the Logo Usage Guidelines are strictly observed.

a copy of the use of the Meshtastic Trademarks is provided to the Trademark Supervisor within seven (7) days of its initial use, for example by sending a URL or other copy of such use.

## **Community Project Logos: M-Powered & M-PWRD**

We are excited to introduce two new logos for our community

members: the "M-Powered" and "M-PWRD" logos. These logos are designed to be used by projects that are powered by Meshtastic technology, offering an easy way to indicate compatibility with our platform.

## Usage Guidelines

### **Variants:**

"M-Powered" with the full word "powered."

"M-PWRD" with an abbreviated "PWRD" for a cleaner, concise design. This version is particularly recommended for 3D printed objects such as enclosures.

**Freedom of Use:** Unlike the main Meshtastic logo, using the "M-Powered" or "M-PWRD" logos does not require a trademark grant. These logos are intended for broader use in the community to signify that a project is Meshtastic-compatible.

**Appropriate Contexts:** In most scenarios, the "M-Powered" or "M-PWRD" logos will be more suitable than the official Meshtastic logo. They are ideal for project documentation, promotional materials, and product designs to show compatibility with Meshtastic.

**Non-Endorsement:** It's important to note that using these logos does not imply endorsement or sponsorship by the Meshtastic Project. They are purely for indicating compatibility or association with Meshtastic technology.

We have provided design files for these logos in our Meshtastic

Design repository.

# **Usage that Require Prior Written Approval**

## **Social Media**

In the past, community members have inquired whether it is permissible to use the Meshtastic Trademarks, including the term "Meshtastic" in a Social Media account. It is not permissible without written permission of Meshtastic.

## **Domain names**

In the past, community members have inquired whether it is permissible to use the Meshtastic Trademarks, including the term "Meshtastic" in an Internet domain name. It is not permissible without written permission of Meshtastic.

## **Non-software goods**

In the past, community members have inquired whether it is permissible to use Meshtastic Trademarks on non-software goods such as Embedded Systems, Enclosures, Hardware, T-shirts, stickers, and pens.

Community members may request from the Trademark Supervisor

designated by the Admins of Meshtastic a license to use the Meshtastic Trademarks on non-software related goods or services, by enquiring at trademark@meshtastic.org. The Trademark Supervisor will be responsible for reviewing samples of the goods and services and managing the relationship.

The adding our logo or other marks to a non-software good may be allowed, with written permission from the the Meshtastic Trademark Supervisor, provided that the non-software good is a 100% original design or the requester has evidence of a grant of use from copyright holder of the original design.

The adding our logo or other marks to a non-software good is not an endorsement or sponsorship of that good by the Meshtastic Project.

## **Unapproved Use**

The following uses of the Meshtastic Trademarks are not approved under any foreseeable circumstances.

Violations of the Logo Usage Guidelines or Trademark Usage Guidelines.

Any use outside these guidelines not by explicit written permission.

Except as set forth herein, Meshtastic retains and reserves all rights to the Meshtastic Trademarks and their use, including the right to modify these guidelines.

These guidelines (except for trademark licenses executed for non software goods) may be amended from time to time at the discretion of Meshtastic and such changes will be effective ten (10) days after the changes are posted. Meshtastic may provide you with notice of such changes, but need not do so. You are responsible for checking this site for any changes.

## **Proper Trademark Use**

One of the purposes of Meshtastic is to encourage the use of Meshtastic software to enable the public to come to trust the use of Meshtastic Trademarks. To achieve this purpose it is important that Meshtastic can quickly be identified. Meshtastic has chosen the Meshtastic Logo to identify such software and is granting usage rights in the Meshtastic Logo (and the other Meshtastic Trademarks) as previously described in this document in order to assure widespread availability.

## **Trademark Usage Guidelines**

### **Guideline**

### **Examples**

When using the Meshtastic Trademarks you must provide the proper trademark symbols and a trademark attribution statement.

Acceptable: Use for the first instance of the Meshtastic Logo

include the ® mark, and include the statement "The Meshtastic logo trademark is the trademark of Meshtastic LLC."

Unacceptable: Never using the ® mark for Meshtastic Logo, nor a trademark statement per the guidelines.

Always distinguish trademarks from surrounding text with at least initial capital letters or in all capital letters.

Never pluralize a trademark. Never use "a" or "the" to refer to an instance of the trademark. Always use a trademark as an adjective modifying a noun, or as a singular noun.

Never use a trademark as a possessive. Instead, the following noun should be used in possessive form or the sentence reworded so there is no possessive.

Never translate a trademark into another language.

Never alter a trademark in any way including through unapproved fonts or visual identifiers.

Never use or register any trademarks that are confusingly similar to, or a play on, the Meshtastic or Meshtastic Logo.

Never combine your company name with the Meshtastic name or use the Meshtastic name in a way that it could be perceived that Meshtastic and your company have an organizational link such as a joint venture.

Never use the Meshtastic Trademarks in a disparaging manner or that violates any federal, state, or international law.

Unacceptable: Cartoon character micturating on the Meshtastic

Logo, applying Meshtastic Logo outside permitted uses.

Never use terminology that states or implies that Meshtastic assumes any responsibility for the performance of your products and services.

The Meshtastic Logo must be hyperlinked to <http://meshtastic.org/>, in contexts where such a hyperlink is technically feasible.

Acceptable: Hyperlinking the Meshtastic Logo where feasible.

Unacceptable: Not hyperlinking the Meshtastic Logo where feasible.

Except as prohibited by law, the person or entity who is using the Meshtastic Trademark under the terms of these Guidelines ("User") acknowledges that Meshtastic is the sole and exclusive owner of the Meshtastic Trademarks and agrees that it will do nothing inconsistent with such ownership either during the term of such use or afterwards. Specifically, User will take no action that will interfere with or diminish Meshtastic's right in the Meshtastic Trademarks. The User acknowledges that the Meshtastic Trademarks are valid under the applicable law and that User's utilization of the Meshtastic Trademarks will not create any right, title or interest in the Meshtastic Trademarks. The User agrees not to apply or assist any third party to register the Meshtastic Trademarks or a confusingly similar designation anywhere in the world. If any application for registration is or has been filed by or

on behalf of User in any country and relates to any mark which, in the reasonable opinion of Meshtastic, is confusingly similar, deceptive or misleading with respect to, or dilutes or any way damages the Meshtastic Trademark, User shall, at Meshtastic's request, abandon all use of such mark, and any registration or application for registration thereof and shall reimburse Meshtastic for all costs and expenses of any opposition or related legal proceeding, including attorneys' fees, instigation by Meshtastic or its authorized representative. Upon the written statement that the User is not using the Meshtastic Trademarks in accordance with the Guidelines, within ten (10) days, User shall modify its use to comply with the Guidelines or cease using the Meshtastic Trademarks.

## **Logo Usage Guidelines**

The Meshtastic logo is a trademark of Meshtastic LLC. In order to protect and grow the Meshtastic brand, we have a distinguishable logo. When displaying the Meshtastic logo, please follow our standard Trademark Guidelines. Other sizes and resolutions of the logo, some suitable for print, can be found [here](#).

To join the discussion about Meshtastic Trademark policies, and participate in shaping future policy visit the [Meshtastic Discourse](#) or [Meshtastic Discord](#).

# Trademark Grants

## Overview

A trademark grant from Meshtastic LLC is an authorization given to individuals, organizations, or projects to use the Meshtastic® trademark and associated logos. This grant is provided under specific terms and conditions that align with Meshtastic's policies. It allows the grantee to legally use the Meshtastic® trademark in connection with their products, services, or projects, indicating compatibility or association with Meshtastic technology. The grant is typically non-exclusive and time-bound, ensuring that the usage adheres to the quality and standards set by Meshtastic LLC while fostering community engagement and innovation. For detailed guidelines and conditions of our trademark policy, please visit our Licensing and Trademark page.

## How to Request a Trademark Grant

If you are interested in obtaining a trademark grant for using the Meshtastic® trademark and logos, please follow these steps:

**Prepare Your Request:** Review the currently approved trademark grants on this page to understand the typical format and information required. Read <https://meshtastic.org/docs/legal/>

licensing-and-trademark/ and follow the process to contact trademark@meshtastic.org for review.

**Edit the Page:** At the bottom of this page, click "Edit this page." This action will redirect you to GitHub, where the page is hosted. Note: A GitHub account is required to make edits. If you don't have one, you will need to create it.

**Add Your Information:** Once on GitHub, use the existing grants as a template to add your information. Include details such as your name or organization, the nature of your project and the agreement from your contact of the Meshtastic Trademark Supervisor.

**Submit for Review:** After adding your information, submit your edit as a pull request. This request will be reviewed by one of the Meshtastic project admins.

**Approval Process:** The admin team will review your submission to ensure it aligns with our trademark policies and agreements with the Trademark Supervisor. If approved, your grant will be listed on this page.

This process ensures transparency and community involvement in the granting of trademark usage, while also maintaining the integrity and purpose of the Meshtastic brand.

# Active Trademark Grants

Grant: <http://meshtastic.pt>

Details: Meshtastic.pt is a fan page created before the trademark usage guidelines were authored. Agreement with Sérgio Matos that meshtastic.pt may continue use of meshtastic in domain name provided they stay non commercial and maintain that they are non-official. The grant is revokable at any time for any reason.

Grant: <https://meshbrasil.com>

Details: Meshbrasil.com is an online shop for Meshtastic powered devices and accessories which carry the "Powered by Meshtastic" logo. The use of the Meshtastic Logo and Trademarks does not imply Meshbrasil.com is sponsored or endorsed by Meshtastic. Meshbrasil.com also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Garth Vander Houwen

Details: Garth is a member of the Meshtastic LLC, is the developer of the iOS app and runs an online shop for Meshtastic powered devices which carry the "Powered by Meshtastic" and "Chirpy" logo. The use of the Meshtastic Logo and Trademarks does not imply Garth is sponsored or endorsed by Meshtastic. Garth also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Anthony (Tony) Good

Details: Tony is an admin and contributer of computer aided design (CAD)/3D designs primarily for device enclosures and accessories, and runs an online shop for Meshtastic powered devices which carry the "Meshtastic" and "M" logos. The use of the Meshtastic Logo and Trademarks does not imply Tony is sponsored or endorsed by Meshtastic. Tony also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: <http://k9rocket.tech>

Details: K9 Rocket Technologies is an open technology development company selling and implementing Meshtastic-powered devices. The devices and their respective promotional content carry the "Powered by Meshtastic", "Meshtastic", & "M" logos. The use of the Meshtastic Logo and Trademarks does not imply K9 Rocket Technologies is sponsored or endorsed by Meshtastic. K9 Rocket Technologies also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason

Grant: Ben Lipsey

Details: Ben Lipsey is a Meshtastic Contributor and Promotional Materials Distributor. Promotional materials carry the "Meshtastic" and "M" logos. Materials also carry the Meshtastic.org URL. The use of the Meshtastic Logo and Trademarks does not imply Ben Lipsey is sponsored or endorsed by Meshtastic. Ben Lipsey also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Mark Birss

Details: Mark Birss is a Meshtastic contributor/developer of DIY devices that carry the "Meshtastic" , Meshtastic.org URL and "M" logos. The use of the Meshtastic Logo and Trademarks does not imply Mark Birss is sponsored or endorsed by Meshtastic. Mark Birss also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant Paul Carney

Details: Paul primarily designs enclosures and assembles complete Meshtastic Radios for sale using modules from TTGO, Heltec and RAK. He runs an online shop for Meshtastic powered devices which carry the "Meshtastic" and M logos. The use of the Meshtastic Logo and Trademarks does not imply Paul is sponsored or endorsed by Meshtastic. Paul also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Keith Monaghan

Details: Keith is a contributer of computer aided design (CAD)/3D designs primarily for device enclosures and accessories, and runs an online shop for Meshtastic powered devices which carry the "Meshtastic" and "M" logos. The use of the Meshtastic Logo and Trademarks does not imply Keith is sponsored or endorsed by Meshtastic. Keith also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Neil Hao

Details: Neil is a contributer of hardware designs, and runs an online shop for Meshtastic powered devices which carry the

"Meshtastic" and "M" logos. The use of the Meshtastic Logo and Trademarks does not imply Neil is sponsored or endorsed by Meshtastic. Neil also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Emmett Plant

Details: Emmett is producing 'Axanar!', a non-commercial Star Trek fan puppet show. The Meshtastic logo/trademark will used only in a positive manner, to hint to the audience that Meshtastic is still up and running in the year 2380. The use of the Meshtastic Logo and Trademarks does not imply Emmett Plant or Axanar! is sponsored or endorsed by Meshtastic. Emmett also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

Grant: Simon - muzi.works

Details: Simon is a Meshtastic designer of devices, device enclosures and accessories that carry the "Meshtastic" , Meshtastic.org URL and "M" logos. The use of the Meshtastic Logo and Trademarks does not imply Simon is sponsored or endorsed by Meshtastic. Simon also agrees to maintain compliance with the Meshtastic Legal requirements. This grant is revokable at any time for any reason.

# Glossary of Terms

## **App / Application / Client Application**

An application that connects to a Meshtastic node, typically for the purpose of sending or receiving data through the mesh network.

## **Band**

A range of frequencies used for LoRa, dependent on region.

Meshtastic further divides these bands into channels. Sometimes identified by the lower and upper bounds of the range (e.g. 902-928MHz), sometimes identified by a center frequency within the range (e.g. 915MHz), and sometimes only by the region they apply to (e.g. US).

## **Broadcast**

Sending a message or data from one device to all other devices within range in the Meshtastic network, rather than to a specific recipient.

## **Channel | Configuration | Frequency Calculator**

At least two definitions in Meshtastic usage: 1) One of 8 configurable channels in the firmware, each supporting a separate name and encryption, with one set as primary and the rest secondary. 2) A specific frequency within a LoRa band that a device can be configured to use.

## **CLI | Guide**

Command Line Interface, a text-based interface used for interacting with software or devices like Meshtastic.

## **Client**

A device or application that connects to a Meshtastic node, typically for the purpose of sending or receiving data through the mesh network.

## **Device**

A physical piece of hardware that utilizes the Meshtastic software and LoRa (Long Range) radio technology to create a decentralized, long-range mesh network.

## **DFU**

Device Firmware Update, a state which a device is placed into for it to receive a firmware update

## **ESP32 | Drivers | Firmware**

A chipset of microcontroller made/designed by Espressif, used by a number of devices. Higher power usage than NRF52, but often cheaper and supports WiFi if desired.

## **Firmware | Guide**

The low-level software programmed onto a Meshtastic device, controlling its hardware functions and enabling it to communicate within the mesh network using LoRa technology. Firmware

## **Flash/Flashing | Guide**

The process of updating or installing firmware on a Meshtastic device. This is typically done using a computer to load new firmware versions or custom software to enhance or modify device functionality.

## **GPIO**

General Purpose Input/Output. An uncommitted digital signal pin on a device

## **LoRa**

A low-power, long-range wireless communication technology used by Meshtastic devices to enable communication over distances of several kilometers without the need for cellular, Wi-Fi, or other traditional network infrastructures.

## **LoS**

Line of Sight, a pathway through only air between two points.

## **Mesh | Algorithm**

In the context of Meshtastic and networking, a mesh refers to a network topology where devices (nodes) are interconnected, allowing them to directly and dynamically communicate with each other. This setup enables data to be relayed across the network, improving coverage and reliability, especially in challenging environments.

## **Message**

A piece of data or text sent between Meshtastic devices over the mesh network, which can include text communications, GPS location updates, and other small data payloads.

## **MHz**

Megahertz, a unit of frequency equal to one million hertz (cycles per second), used to specify the operating frequency of LoRa devices in the Meshtastic network, affecting range and data rate.

## **Module | Software Modules | Hardware Modules**

At least two definitions in Meshtastic usage: 1) A software plug-in to expand the capabilities of a Meshtastic device. 2) A hardware component or add-on for a Meshtastic device, such as a temperature sensor or GPS.

## **MQTT**

An acronym for Message Queuing Telemetry Transport, is a lightweight messaging protocol designed for small sensors and mobile devices, enabling efficient data transmission in the Meshtastic network for Internet connectivity and integration with IoT platforms. See <https://en.wikipedia.org/wiki/MQTT>. In Meshtastic, MQTT is used to connect a node to the internet, and can be used to connect multiple meshes to each other.

## **Node**

A unit within the Meshtastic network that can send, receive, and relay messages, helping to form and extend the mesh network's coverage.

## **NRF52 | Drivers | Firmware**

A microcontroller chipset made by Nordic, used by a number of devices used by several devices such as the RAK Meshtastic Starter Kit and the Lilygo T-Echo. Lower power usage than ESP32.

## **Packet**

A formatted unit of data sent over the network. In Meshtastic, packets carry messages, GPS locations, and other information through the LoRa mesh network.

## **Protobuf | Reference**

Protocol Buffers, a method developed by Google for serializing structured data, used in Meshtastic for efficient communication protocol between devices.

## **PSK | Encryption**

Pre-Shared Key, a secret code or passphrase used in Meshtastic channels for encryption, ensuring that only devices with the

matching PSK can communicate within that specific channel.

### **Repeater | Role Definitions**

Infrastructure node for extending network coverage by relaying messages with minimal overhead. Not visible in Nodes list.

### **Router | Role Definitions**

Infrastructure node for extending network coverage by relaying messages. Visible in Nodes list.

### **rp2040**

A microcontroller chip developed by Raspberry Pi, featuring dual ARM Cortex-M0+ processors.

### **RX**

Abbreviation for Receive.

### **Sensor**

A device component that detects and responds to some type of input from the physical environment. In Meshtastic, sensors can be used to gather environmental data (e.g., temperature, humidity, GPS location) which can then be transmitted over the mesh network for monitoring or other applications.

### **Serial**

A communication protocol used for the transmission of data between the Meshtastic device and a computer or other devices. Typically over USB or UART.

### **SNR**

Signal-to-Noise Ratio, a measure used in communications to quantify the level of a desired signal to the level of background noise. In Meshtastic and other wireless systems, a higher SNR indicates a clearer signal that can enhance the reliability and

quality of data transmission.

## **SWR**

Standing Wave Ratio, a measure of the efficiency of the radio frequency (RF) power transmission from a transmitter through a transmission line to an antenna in Meshtastic devices. It indicates the ratio of the amplitude of a standing wave at maximum to the amplitude at minimum, with an ideal SWR close to 1:1, signifying that most of the power is transmitted to the antenna with minimal reflections.

## **Telemetry**

The sending of sensor data or system metrics over the mesh network.

## **Tranceiver**

A device capable of both transmitting and receiving communications.

## **Transmit**

The act of sending data, such as messages or GPS locations, from one Meshtastic device to another over the LoRa network.

## **TX**

Abbreviation for Transmit.