Getting Started with the Meshtastic Python Library

The Meshtastic Python library (`meshtastic` on PyPI) provides a clean API for connecting to Meshtastic devices, reading their state, sending messages, and reacting to received packets via callbacks. This page covers installation, all three connection methods (serial, TCP, BLE), and complete working code examples for the most common operations.

Installation

```
# Core library (serial + TCP)
pip install meshtastic

# With BLE support
pip install "meshtastic[ble]"

# Development / latest from GitHub
pip install git+https://github.com/meshtastic/python.git
```

The library requires Python 3.9+. Its core dependencies are `protobuf`, `pyserial`, and `pypubsub`. BLE support additionally requires `bleak`.

Connecting via Serial

```
import meshtastic
import meshtastic.serial_interface

# Auto-detect first available Meshtastic device
iface = meshtastic.serial_interface.SerialInterface()

# Specify a port explicitly
iface = meshtastic.serial_interface.SerialInterface(devPath="/dev/ttyUSB0")
iface = meshtastic.serial_interface.SerialInterface(devPath="COM4") # Windows
```

When a `SerialInterface` is created, the library:

1. Opens the serial port at 115200 baud.
2. Performs a handshake to confirm the device is running Meshtastic firmware.
3. Downloads the full node database and device configuration from the node.
4. Sets the node's RTC to the host system time.

The constructor blocks until the download completes (typically 1 - 3 seconds). After that the `iface` object is fully populated.

Connecting via TCP

```
import meshtastic.tcp_interface

# Connect to a node by IP address
iface = meshtastic.tcp_interface.TCPInterface(hostname="192.168.1.42")

# Connect by mDNS hostname
iface = meshtastic.tcp_interface.TCPInterface(hostname="meshtastic.local")
```

TCP connection is useful for nodes deployed without USB access. The default port is 4403; you can override it with the `portNumber` parameter. The TCP interface requires the node to have Wi-Fi enabled and the TCP API enabled in Radio Configuration.

Connecting via BLE

```
import asyncio
import meshtastic.ble_interface

# Scan for available BLE devices
async def scan():
 devices = await meshtastic.ble_interface.BLEInterface.scan()
 for d in devices:
 print(d.name, d.address)

asyncio.run(scan())

# Connect by device name or MAC address
iface = meshtastic.ble_interface.BLEInterface("Meshtastic_abcd")
```

Reading Device Info

```
import meshtastic
import meshtastic.serial_interface
import json

iface = meshtastic.serial_interface.SerialInterface()

# My node info
my_info = iface.getMyNodeInfo()
print("My node number:", my_info["num"])
print("My user:", my_info.get("user", {}).get("longName"))

# Firmware metadata
meta = iface.metadata
if meta:
 print("Firmware:", meta.firmware_version)
 print("Hardware:", meta.hw_model)

# Full local config as a protobuf object
config = iface.localConfig
print("Hop limit:", config.lora.hop_limit)
print("Region:", config.lora.region)

# Channel config
for ch in iface.localChannels:
 print(f"Channel {ch.index}: role={ch.role}, name={ch.settings.name or 'default'}")

iface.close()
```

Listing Nodes

```
import meshtastic
import meshtastic.serial_interface
import time

iface = meshtastic.serial_interface.SerialInterface()
```

```
nodes = iface.nodes # dict keyed by "!<hex_node_id>"

for node_id, node in nodes.items():
 user = node.get("user", {})
 pos = node.get("position", {})
 metrics = node.get("deviceMetrics", {})
 last_heard = node.get("lastHeard", 0)
 age_minutes = (time.time() - last_heard) / 60 if last_heard else None

 print(f"Node: {node_id}")
 print(f" Name: {user.get('longName', 'Unknown')}")
 print(f" Short name: {user.get('shortName', '???')}")
 print(f" Hardware: {user.get('hwModel', 'Unknown')}")
 if pos.get("latitudeI"):
 lat = pos["latitudeI"] / 1e7
 lon = pos["longitudeI"] / 1e7
 print(f" Position: {lat:.5f}, {lon:.5f} alt={pos.get('altitude', 0)}m")
 if metrics:
 print(f" Battery: {metrics.get('batteryLevel', '?')}%")
 print(f" Chan util: {metrics.get('channelUtilization', '?')}%")
 print(f" SNR: {node.get('snr', 'N/A')} dB")
 if age_minutes is not None:
 print(f" Last heard: {age_minutes:.1f} min ago")
 print()

iface.close()
```

Sending a Text Message

```
import meshtastic
import meshtastic.serial_interface

iface = meshtastic.serial_interface.SerialInterface()

# Broadcast to all nodes on the primary channel
iface.sendText("Hello mesh!")
```

```
# Send to a specific node (direct message)
iface.sendText("Hello from Python", destinationId="!aabbccdd")

# Send on a secondary channel (index 1)
iface.sendText("Private message", channelIndex=1)

# Send and wait for acknowledgement
import time
iface.sendText("Test with ACK", wantAck=True)
time.sleep(5) # allow time for ACK to arrive

iface.close()
```

The `sendText` method returns immediately after queuing the packet. For confirmed delivery you must listen for the ACK packet via a callback (see below).

Receiving Messages with Callbacks

The library uses `pypubsub` for its event system. Subscribe to topics before creating the interface, or add subscriptions after connection. The main topics are:

- `meshtastic.receive` - all decoded packets from the mesh.
- `meshtastic.receive.text` - text message packets only.
- `meshtastic.receive.position` - position packets.
- `meshtastic.receive.telemetry` - telemetry packets.
- `meshtastic.receive.user` - node-info (user) packets.
- `meshtastic.connection.established` - fired when connection and download complete.
- `meshtastic.connection.lost` - fired on disconnect.

```
import meshtastic
import meshtastic.serial_interface
from pubsub import pub
import time

def on_connect(interface, topic=pub.AUTO_TOPIC):
 # Called when the library finishes downloading node data.
 print(f"Connected! Nodes in mesh: {len(interface.nodes)}")

def on_receive(packet, interface):
 # Called for every decoded packet from the mesh.
```

```
 port_num = packet.get("decoded", {}).get("portnum", "UNKNOWN")
 from_id = packet.get("fromId", "?")
 to_id = packet.get("toId", "?")
 hops = packet.get("hopStart", 0) - packet.get("hopLimit", 0)

 if port_num == "TEXT_MESSAGE_APP":
 text = packet["decoded"].get("text", "")
 print(f"[TEXT] {from_id} -> {to_id} ({hops} hop(s)): {text}")

 elif port_num == "POSITION_APP":
 pos = packet["decoded"].get("position", {})
 lat = pos.get("latitudeI", 0) / 1e7
 lon = pos.get("longitudeI", 0) / 1e7
 print(f"[POSITION] {from_id}: {lat:.5f}, {lon:.5f}")

 elif port_num == "TELEMETRY_APP":
 tel = packet["decoded"].get("telemetry", {})
 dm = tel.get("deviceMetrics", {})
 print(f"[TELEMETRY] {from_id}: battery={dm.get('batteryLevel')}%, "
 f"chan_util={dm.get('channelUtilization'):.1f}%")

def on_text_receive(packet, interface):
 # Called only for TEXT_MESSAGE_APP packets.
 text = packet.get("decoded", {}).get("text", "")
 print(f"Text message received: {text!r}")

pub.subscribe(on_connect, "meshtastic.connection.established")
pub.subscribe(on_receive, "meshtastic.receive")
pub.subscribe(on_text_receive, "meshtastic.receive.text")

iface = meshtastic.serial_interface.SerialInterface()

try:
 print("Listening for packets. Press Ctrl-C to exit.")
 while True:
 time.sleep(1)
except KeyboardInterrupt:
 pass
finally:
```

```
 iface.close()
```

Complete Minimal Example: Echo Bot

```
# echo_bot.py -- Meshtastic echo bot
# Responds to any text message with "Echo: <original message>"
import meshtastic
import meshtastic.serial_interface
from pubsub import pub
import time

iface = None

def on_receive(packet, interface):
 decoded = packet.get("decoded", {})
 if decoded.get("portnum") == "TEXT_MESSAGE_APP":
 text = decoded.get("text", "")
 from_id = packet.get("fromId")
 my_id = interface.getMyNodeInfo()["user"]["id"]
 # Don't echo our own messages
 if from_id != my_id:
 interface.sendText(f"Echo: {text}", destinationId=from_id)
 print(f"Echoed to {from_id}: {text!r}")

pub.subscribe(on_receive, "meshtastic.receive")
iface = meshtastic.serial_interface.SerialInterface()
print("Echo bot running. Press Ctrl-C to stop.")
try:
 while True:
 time.sleep(1)
except KeyboardInterrupt:
 pass
finally:
 if iface:
 iface.close()
```

Closing the Interface

Always call `iface.close()` when done. This cleanly shuts down the background receiver thread and closes the serial/TCP/BLE connection. Failing to close the interface can leave the serial port locked, preventing other tools (the app, another script) from connecting.

```
# Best practice: use a try/finally block
iface = meshtastic.serial_interface.SerialInterface()
try:
 # ... your code ...
 pass
finally:
 iface.close()
```

Revision #2
Created 2026-05-03 05:39:39 UTC by Mesh America Admin
Updated 2026-05-03 12:59:14 UTC by Mesh America Admin