

Meshtastic Python Scripting

The meshtastic Python Library

The official **meshtastic** Python library provides programmatic access to any Meshtastic device. It exposes the full device API: sending and receiving messages, reading/writing configuration, querying node lists, requesting telemetry, and subscribing to real-time events.

Install:

```
pip install meshtastic
```

Official documentation: python.meshtastic.org

Source and examples: github.com/meshtastic/python

Connection Types

Interface class	Transport	Typical use
<code>meshtastic.SerialInterface</code>	USB serial (CDC ACM)	Direct connection, most reliable; default port auto-detected or specify <code>/dev/ttyUSB0</code>
<code>meshtastic.TCPInterface</code>	TCP/IP over WiFi	Wireless connection to nodes with WiFi enabled (ESP32 devices); specify host IP (default port 4403)
<code>meshtastic.BLEInterface</code>	Bluetooth Low Energy	Short-range wireless; requires BlueZ on Linux or appropriate BLE stack

Basic Usage Examples

The snippets below assume you have already created an `iface` object as shown in the first example. When copying a single snippet, include that connection setup line first.

Connect and print node info

```
import meshtastic
import meshtastic.serial_interface

iface = meshtastic.serial_interface.SerialInterface()
print(iface.nodes) # dict of all known nodes
iface.close()
```

Send a message to a channel

```
iface.sendText("Hello mesh", channelIndex=0)
# channelIndex=0 is the primary channel; 1 - 7 are secondary channels
```

Subscribe to received messages

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

def on_receive(packet, interface):
    print(f"Received: {packet}")

iface = meshtastic.serial_interface.SerialInterface()
pub.subscribe(on_receive, "meshtastic.receive")

# Keep the script alive while the interface thread processes events
import time
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    iface.close()
```

Get node list with last-heard timestamps

```
for node_id, node in iface.nodes.items():
    last_heard = node.get("lastHeard", "unknown")
    name = node.get("user", {}).get("longName", node_id)
    print(f"{name}: last heard {last_heard}")
```

Read and write device config

```
local = iface.getNode('^local')
# Read
print(local.localConfig.lora.hop_limit)
# Write
local.localConfig.lora.hop_limit = 5
local.writeConfig("lora")
```

Note: `getNode('^local')` returns the local Node object. `localConfig`, `writeConfig()`, `writeChannel()`, `getMyNodeNum()` and `removeNode()` are methods/attributes of the **Node** object (reached via `getNode(...)`), not of the base interface.

Request position from local node

```
iface.getNode('^local').requestPosition()
```

To set a fixed position programmatically (including altitude), use

```
iface.localNode.setFixedPosition(lat, lon, alt).
```

Automation Use Cases

- **Automated mesh announcements** - scheduled weather, news, or system status broadcasts on a channel.
- **Message bridges** - relay messages between the mesh and Telegram, Discord, Matrix, or SMS (via Twilio). Bidirectional bridging is straightforward with the pub/sub event model.
- **Telemetry logging** - write incoming telemetry packets to CSV, SQLite, or InfluxDB for Grafana dashboards.
- **Position mapping** - forward GPS positions to a self-hosted map (e.g. Traccar, OwnTracks, or a custom Leaflet map).
- **Alert systems** - trigger SMS or email alerts when specific nodes go offline (last-heard threshold exceeded).
- **Config management** - script bulk configuration changes across many nodes connected via TCP.

Privacy note: the bridging and position-mapping patterns export mesh content and node positions to external services. Make sure that exposure is acceptable for your deployment before forwarding messages or locations off the RF mesh.

Common Patterns

```
# Connect (serial, auto-detect port)
iface = meshtastic.serial_interface.SerialInterface()

# Connect (TCP)
iface = meshtastic.tcp_interface.TCPInterface("192.168.1.50")

# Send text
iface.sendText("Hello mesh", channelIndex=0)

# Send to specific node (DM)
iface.sendText("Private message", destinationId="!a1b2c3d4", channelIndex=0)

# Request position
iface.getNode('^local').requestPosition()

# Write channel config
ch = iface.getNode('^local').channels[0]
ch.settings.name = "MyNet"
iface.getNode('^local').writeChannel(0)
```

Event Loop and Threading

The library spawns a background thread that reads from the serial/TCP connection and dispatches events via the **PyPubSub** pub/sub system. Your `on_receive` callback is called in that background thread - use thread-safe data structures (queues, locks) if you share state with your main thread.

Available topics:

- `meshtastic.receive` - any incoming packet
- `meshtastic.receive.text` - text messages only
- `meshtastic.receive.position` - position updates
- `meshtastic.receive.user` - node user/info packets
- `meshtastic.receive.data.portnum` - data packets by port number (telemetry arrives here under the `TELEMETRY_APP` portnum; there is no separate `meshtastic.receive.telemetry` topic)
- `meshtastic.connection.established` - fired after successful connect
- `meshtastic.connection.lost` - fired on disconnect

Error Handling

- **Serial port busy:** only one process can hold the serial port. Close the [Meshtastic app](#) or CLI before running your script. Use `try/finally: iface.close()` to release cleanly.
- **Reconnection on disconnect:** subscribe to `meshtastic.connection.lost` and re-instantiate the interface with exponential backoff.

- **Timeout handling:** `SerialInterface` has a `noProto` parameter - set it to `True` for raw serial access without waiting for a Meshtastic handshake, useful for debugging hardware.
 - **Packet validation:** always guard against missing keys in the packet dict; not all fields are present in every packet type.
-

Revision #5

Created 2026-05-03 04:10:58 UTC by Mesh America Admin

Updated 2026-06-09 12:09:39 UTC by Mesh America Admin