

Node-RED Flows for Mesh Automation

Overview

Node-RED is a visual flow-based programming tool that acts as powerful middleware between your Meshtastic MQTT feed and virtually any other service. It runs on Linux (including Raspberry Pi), inside Home Assistant, or on any Node.js-capable machine.

About the JSON MQTT topics used below: Meshtastic publishes decoded JSON packets to `msh/REGION/2/json/CHANNELNAME/USERID` - the segment after `json` is the **channel name**, not the packet type. There is no `.../json/text/` sub-topic. To catch all channels, subscribe to `msh/US/2/json/#` and filter on the packet type (a `type` field *inside* the JSON, e.g. `"text"`, `"position"`) in a switch/function node. Also note these JSON flows require the gateway to publish **decoded (plaintext)** packets to the broker - which bypasses channel encryption at the broker. If that broker is the public one, the decoded content (including any "emergency" channel) is public. Use a private, access-controlled broker for any sensitive channel.

Installing Node-RED

Standalone on Raspberry Pi / Linux:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
sudo systemctl enable nodered
sudo systemctl start nodered
```

Access the editor at `http://your-pi-ip:1880`.

In Home Assistant: Install the *Node-RED* add-on from the add-on store. It integrates directly with your HA entities.

Core Flow Pattern 1: Message Logger

This flow captures all text messages from Meshtastic and writes them to a log file.

```
[MQTT In] → [JSON Parse] → [Switch: msg.payload.type == "text"] → [Function: format line] → [File Write]
```

MQTT In topic: msh/US/2/json/# (subscribe to all channels; filter type in the switch)

Function node:

```
msg.payload = new Date().toISOString() + " [" +  
msg.payload.from.toString(16).toUpperCase() + "]" +  
msg.payload.payload + "  
";  
return msg;  
File node: /home/pi/mesh_log.txt (append mode)
```

In the decoded text-message JSON, `from` is a top-level decimal node ID (prefix it with `!` when displaying the hex), and for a text packet the message string is at `payload.payload` (not `payload.payload.text`). Verify the exact field nesting against a live JSON sample, since it varies by packet type.

Core Flow Pattern 2: Position Tracker to Google Sheets

Filter position packets by a specific node ID and push lat/lon to a Google Sheet via the Sheets API node (`node-red-contrib-google-sheets`).

```
[MQTT In: msh/US/2/json/#] → [JSON Parse] → [Switch: msg.payload.type == "position" AND  
msg.payload.from == targetNodeId]  
→ [Function: build row] → [Google Sheets: append row]
```

Function node:

```
var pos = msg.payload.payload;
```

```
msg.payload = [  
  new Date().toISOString(),  
  pos.latitude_i / 1e7,  
  pos.longitude_i / 1e7,  
  pos.altitude  
];  
return msg;
```

Core Flow Pattern 3: Two-Way Bridge - Meshtastic ↔ Telegram

Install `node-red-contrib-telegrambot`. This flow forwards incoming mesh messages to a Telegram chat and relays Telegram replies back to the mesh channel.

Meshtastic → Telegram:

```
[MQTT In: msh/US/2/json/#] → [JSON] → [Switch: type == "text"] → [Function: build TG msg]  
→ [Telegram Sender]
```

Telegram → Meshtastic:

```
[Telegram Receiver] → [Function: build MQTT payload]  
→ [MQTT Out: msh/US/2/json/mqtt/] (downlink topic)
```

Downlink is more involved than it looks. Publishing arbitrary JSON to an output topic does *not* reliably inject a message into the mesh. For the firmware to transmit a downlinked message the gateway must have downlink enabled for that channel (`mqtt` module / channel `downlink_enabled`), and the payload must use the correct topic and envelope that the firmware accepts. If those conditions are not met, your Telegram replies will silently never reach the mesh. Confirm the current downlink topic/envelope against the Meshtastic MQTT integration docs before relying on this direction.

Emergency Channel SMS Forwarding via Twilio

Install `node-red-contrib-twilio`. Watch a dedicated emergency channel (channel index 1, for example) and forward any message to a phone number via SMS:

```
[MQTT In: msh/US/2/json/#] → [JSON] → [Switch: type == "text" AND channel == 1]
→ [Function: format SMS] → [Twilio: send SMS]
```

Function node:

```
var from = msg.payload.from.toString(16).toUpperCase();
msg.payload = "MESH EMERGENCY from !" + from + ": " +
msg.payload.payload;
return msg;
```

Set the Twilio node with your Account SID, Auth Token, and destination phone number.

Caution - this is a best-effort convenience path, not a primary emergency channel. This SMS-forwarding chain depends on the gateway hearing the packet over RF *and* on the gateway's internet/cellular connection, the MQTT broker, Node-RED, and Twilio all being up - any of which can fail silently during the exact grid-down / internet-outage scenario an emergency net plans for. Treat it as a convenience overlay on top of direct RF monitoring, never as your primary emergency communications method.

Exporting Flows

To share or back up a flow, select all nodes with **Ctrl+A**, then go to **Menu → Export → Clipboard**. The resulting JSON can be imported on any Node-RED instance. Store your flow exports in version control alongside your Meshtastic configuration.

Revision #3

Created 2026-05-03 06:43:47 UTC by Mesh America Admin

Updated 2026-06-09 12:09:33 UTC by Mesh America Admin