

Role Configuration and Tuning

Deep-dive configuration of ROUTER, ROUTER_CLIENT, and REPEATER roles including hop limit tuning and fixed-position setup.

- [ROUTER vs ROUTER_CLIENT vs REPEATER Role: Deep Dive](#)
- [Hop Limit Configuration for Repeaters](#)
- [Fixed Position for Repeater Nodes](#)

ROUTER vs ROUTER_CLIENT vs REPEATER Role: Deep Dive

Meshtastic provides several device roles for infrastructure nodes that exist to extend network reach rather than serve end users. On current firmware the relevant roles are **ROUTER**, **ROUTER_LATE**, and **REPEATER** (with **CLIENT** being the right choice for the overwhelming majority of ordinary nodes, since CLIENT nodes already rebroadcast via managed flooding). Choosing the wrong role wastes resources, creates unnecessary air-time, or silently breaks the capabilities operators expect. This page dissects each role at the firmware level so you can make an informed decision for every node you deploy.

Important deprecation notice. The **ROUTER_CLIENT** role referenced in this page's title was **retired in firmware 2.3.15** and is no longer a selectable `device.role` value. Do not attempt to set it. If you need a node that both relays and is used interactively by its operator, use **CLIENT** (which already relays) or **ROUTER**. Separately, the **REPEATER** role has itself been **deprecated as of firmware ~2.7.x**; for new infrastructure prefer **ROUTER** or **ROUTER_LATE**. The valid current `device.role` values are CLIENT, CLIENT_MUTE, CLIENT_HIDDEN, ROUTER, ROUTER_LATE, REPEATER, TRACKER, SENSOR, TAK, TAK_TRACKER, and LOST_AND_FOUND. The historical ROUTER_CLIENT material below is retained for context only and clearly marked as legacy.

The Infrastructure Roles at a Glance

Capability	ROUTER	ROUTER_LATE	REPEATER (deprecated)
Rebroadcasts received packets (once, with higher priority than CLIENT roles)	Yes	Yes (rebroadcasts last, after other nodes)	Yes
Power-saving sleep behaviour	Forced on automatically (ESP32); cannot be disabled	Configurable via <code>power.is_power_saving</code>	Does not force sleep by default
Visible in the node list / sends NodeInfo	Yes	Yes	No (anonymous relay, sends no NodeInfo)

Capability	ROUTER	ROUTER_LATE	REPEATER (deprecated)
App connectivity (BLE / WiFi / Serial) by default	Off by default	On (client-style)	Off
Relative overhead	Standard	Standard	Minimal overhead (per docs)

All of ROUTER, ROUTER_LATE, and REPEATER rebroadcast each packet *once* with higher priority than ordinary CLIENT nodes. The retired ROUTER_CLIENT role has been omitted from this table; it is not settable on current firmware.

ROUTER Role

A **ROUTER** node is the canonical fixed-infrastructure role, designed for stationary, well-placed nodes. When you set a node to ROUTER it adjusts its behavior in the following ways:

- ROUTER automatically enables power-saving sleep (ESP32) via `power.is_power_saving`, and this **cannot be turned off**. The LoRa radio stays in standby to wake on incoming packets — it is not a node that ignores power management.
- It defaults BLE / WiFi / Serial connectivity **off**, so you don't normally connect to it to text directly.
- Received packets are rebroadcast once, with higher priority than CLIENT roles, subject to duplicate-detection and the hop counter.
- The device **is visible in the node list** and sends its own `NodeInfo` (and position, if configured) so other nodes know it exists and can display it on the map.
- A ROUTER relays standard packets, including traceroute responses, like any non-mute node. (NeighborInfo is a separate, optional module any role can enable — it is not a defining trait of the ROUTER role.)
- A ROUTER *can* still originate and receive messages at the protocol level. What's true is that its app connectivity (BLE/WiFi/Serial) is off by default, so in current firmware it does not present an interactive messaging UI and you typically won't have an app connected to send from it. This is a connectivity default, not a removed "Send" feature — the firmware does not omit the send code path. If interactive messaging at the site is needed, use CLIENT or ROUTER_LATE rather than ROUTER.

Setting the ROUTER role via CLI

```
meshtastic --set device.role ROUTER
```

Verifying the role was applied

```
meshtastic --get device.role
```

The `--get` verb is the standard counterpart to `--set` for reading any config field in the Python CLI.

ROUTER_LATE Role

ROUTER_LATE is the modern infrastructure role for a node that should rebroadcast *only after* other nodes have had a chance to, rather than with top priority. It is useful within a local cluster where you want a relay that fills gaps without dominating airtime. Unlike ROUTER, it does not force power-saving sleep on, and it keeps client-style app connectivity available. For most stationary backbone infrastructure prefer ROUTER; choose ROUTER_LATE for nodes that should defer to the rest of the mesh first.

ROUTER_CLIENT Role (retired — legacy reference only)

ROUTER_CLIENT was retired in firmware 2.3.15 and is not a settable role on current firmware. Historically it was described as a "superset of ROUTER" that retained ROUTER's relay behaviour while also letting the node originate messages as a client. This is preserved here only for operators encountering older documentation or very old firmware:

- It was described as allowing text messages to be composed and sent from the device. On current firmware, if you need a relay that an operator can also message from, use **CLIENT** (which relays via managed flooding).
- It was described as broadcasting position from the device's own GPS or configured coordinates. That position-broadcast behaviour now applies to CLIENT-family roles.
- It was promoted for an operator who wanted to use the node interactively during events or emergency activations without compromising the relay function. Today, use a **CLIENT** node for that. (Note: Meshtastic carries text, position, and telemetry — it does not carry voice — so "voice-channel communication" was never accurate terminology.)

Do not use the legacy command below on current firmware — it will fail because ROUTER_CLIENT is no longer an accepted `device.role` value. For a relay that an operator also uses interactively, set `device.role CLIENT` instead.

Legacy ROUTER_CLIENT CLI (no longer valid)

```
# Retired in firmware 2.3.15 – this command fails on current firmware.  
# Use:  meshtastic --set device.role CLIENT  
# (legacy:  meshtastic --set device.role ROUTER_CLIENT)
```

REPEATER Role (deprecated)

The **REPEATER** role is the most minimal relay option, designed for nodes that should retransmit packets with minimal overhead and produce no broadcasts of their own. **Note: REPEATER was deprecated as of firmware ~2.7.x; for new infrastructure consider ROUTER or ROUTER_LATE.** Its documented behaviour:

- Received packets are rebroadcast once, with higher priority than CLIENT roles — similar to ROUTER in the act of relaying.
- It is **not visible in the node list** and does *not* send `NodeInfo` — it is effectively anonymous on the mesh. It will not appear in the node list of clients that have not heard it directly.
- It does not originate its own broadcasts (telemetry, position, `NodeInfo`); per the docs it "only responds to other nodes' packets instead of originating messages."
- Per the docs it rebroadcasts with minimal overhead. (It does *not* force power-saving sleep, unlike ROUTER.)

The anonymity of the REPEATER role is a deliberate design choice: relay nodes that aren't individually addressed produce less management overhead on the mesh. Because a REPEATER is not in the node list, you generally cannot select or ping it by node ID in the app the way you can a ROUTER.

Setting the REPEATER role via CLI

```
meshtastic --set device.role REPEATER
```

Hop Count Handling Across Roles

All relaying roles participate in hop-count decrement identically. When a packet arrives with a remaining hop count of N, the forwarding node decrements it to N-1 before rebroadcasting. If N is already 0, the packet is consumed locally but not forwarded. ROUTER and REPEATER have rebroadcast *priority* (they may rebroadcast sooner, and defer less to other rebroadcasters) but they still decrement and honour `hop_limit` — there is no hop-counter bypass or "always forward regardless" mode.

The practical consequence is that placing a relay mid-path does use a hop. Default `hop_limit` is **3** and the maximum is **7**; "really, 3 is fine" per the docs. Ensure the number of relay hops between any two endpoints does not exceed your configured `hop_limit`. Raising `hop_limit` to "fix" dropped messages can worsen congestion (more airtime and collisions), so pair any increase with a check on channel utilisation.

Choosing the Right Role

Community fixed repeater on a hilltop or tower

Use **ROUTER**. The node should be visible to the community (appears in node list, sends NodeInfo, relays traceroutes) but should not generate user traffic of its own. Operators can still access the serial console locally for configuration. For a node that should defer to the rest of the mesh before rebroadcasting, consider **ROUTER_LATE**.

```
meshtastic --set device.role ROUTER
```

Ham operator's home station that also relays

Use **CLIENT**. CLIENT nodes already rebroadcast via managed flooding, so a home station set to CLIENT both relays and lets you participate in mesh conversations — sending alerts, coordinating with your community, or running net check-ins from the same hardware. (The old advice to use ROUTER_CLIENT here is obsolete; ROUTER_CLIENT was retired in 2.3.15.)

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

Minimal-overhead anonymous relay

Historically **REPEATER** was the choice for a small node tucked into a building to bridge two otherwise disconnected areas with no topology visibility. Because REPEATER is deprecated as of firmware ~2.7.x, prefer **ROUTER** (or **ROUTER_LATE**) for new deployments; only use REPEATER if you specifically need an anonymous relay and understand it is being phased out.

```
meshtastic --set device.role ROUTER    # preferred; REPEATER is deprecated
```

Power Consumption Comparison

Idle power profiles differ by role. A **REPEATER** keeps the radio active and does not force sleep, whereas a **ROUTER** uses forced power-saving sleep (ESP32) that cannot be disabled — so their idle draw is not the same. The differences otherwise arise from background processing:

- **REPEATER** has minimal overhead per the official docs, because it skips originating broadcasts and node-presence maintenance. (We do not cite a specific kilobyte heap figure, as no firmware source quantifies it.)
- **ROUTER** forces power-saving sleep on (ESP32); **ROUTER_LATE** leaves sleep configurable via `power.is_power_saving`. If a GPS is active, GPS module current (which varies widely by module — consult the module's datasheet) can dominate other draw.
- LoRa TX current is board- and TX-power-dependent. For an SX1262-class radio alone, TX is roughly ~118 mA at +22 dBm and RX is roughly ~5 mA (Semtech SX1262 datasheet); boards with an external PA draw more, and total board draw (including the MCU) is higher than the radio alone. Treat any single current figure as approximate and board-dependent.
- Power-saving sleep is broadly available via `power.is_power_saving` to most roles (all except TRACKER and SENSOR). It is **not** exclusive to CLIENT: ROUTER forces it on automatically, while REPEATER does not sleep by default.

Quick CLI power-related settings for infrastructure nodes

```
# Disable Bluetooth to save power (figure is approximate; verify against the Bluetooth config docs)
```

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

```
# Reduce the screen on-time to save power. Note: the documented field is
```

```
display.screen_on_secs.
```

```
# A value of 0 is the 10-minute DEFAULT, not "off"; set a small nonzero value to dim sooner.
```

```
meshtastic --set display.screen_on_secs 10
```

```
# Set a lower TX power only if nodes are nearby (reduces TX draw and channel utilisation).
```

```
# tx_power 0 is the default and means "use the region-legal maximum." Never set a higher fixed
```

```
# value than your region/antenna combination legally allows (note the >6 dBi gain-reduction  
rule).
```

```
meshtastic --set lora.tx_power 17
```

Hop Limit Configuration for Repeaters

The hop limit is one of the most important and most misunderstood parameters in a Meshtastic mesh. Setting it correctly reduces unnecessary rebroadcasts, controls how far a message propagates, and prevents broadcast storms that saturate the channel. This page explains exactly what `hop_limit` does, when to change it, and the CLI commands to apply it.

What hop_limit Controls

Every Meshtastic packet carries a **hop count field** in its header. This field is initialised to the `hop_limit` value configured on the *originating* node at the time the packet is sent. Each relay node (ROUTER, ROUTER_LATE, or REPEATER) decrements this field by 1 before rebroadcasting. (The older ROUTER_CLIENT role was deprecated and removed in firmware 2.3.15 — use ROUTER or ROUTER_LATE instead; REPEATER is also deprecated as of firmware ~2.7.x.) When a node receives a packet with a hop count of 0 it delivers the packet locally but does *not* forward it further. (See the Meshtastic mesh algorithm docs: "If any mesh node sees a packet with a HopLimit other than zero, it will decrement that HopLimit and attempt to rebroadcast on behalf of the original sending node.")

In a simple linear chain, the maximum relay chain length is:

```
Maximum relaying nodes = hop_limit
```

```
Nodes reached along an idealised single chain = hop_limit + 1 (originator counts as hop 0)
```

Note: real meshes are not linear chains. Many nodes can hear each hop, so the total number of nodes that actually hear a packet is topology-dependent — the formula above describes an idealised single chain, not a fixed total node count.

With the default `hop_limit = 3`, a packet originating at node A can reach:

- All nodes within direct radio range of A (hop 0 -> 1)
- All nodes reachable via one relay (hop 1 -> 2)
- All nodes reachable via two relays (hop 2 -> 3)
- Once a packet's remaining HopLimit reaches 0 it is delivered locally but not rebroadcast further (hop 3 -> consumed, not forwarded)

In practice the default of 3 covers most community-sized networks with reasonable relay density.

The Default: hop_limit = 3

As of current firmware (2026), the firmware default is 3 — a careful balance. It is high enough to traverse a typical multi-node mesh with a few infrastructure repeaters, and low enough that a single rogue or misconfigured node cannot cause runaway rebroadcasting. (This is a firmware-version-dependent default; verify against your installed firmware.) Leave it at 3 unless you have a specific and measured reason to change it.

When to Increase the Hop Limit

Large geographically spread networks

If your community network spans a large geographic area - for example a county-wide emergency communications mesh with repeaters spaced 20 - 30 km apart - a hop count of 3 may not be sufficient to bridge the entire path. In this case, increasing to 4 or 5 gives messages the relay budget to traverse more infrastructure nodes. Bear in mind that raising hop_limit increases airtime and channel utilisation; prefer adding infrastructure to shorten paths where you can.

Before increasing, first verify the actual hop count needed by running a traceroute between the two furthest nodes:

```
meshtastic --traceroute '!abcd1234'
```

Count the intermediate hops in the output. Set `hop_limit` to at least that number plus one for margin.

```
# Increase hop limit on the originating/infrastructure node
meshtastic --set lora.hop_limit 4
```

When to Decrease the Hop Limit

Small, dense urban networks

In a dense neighbourhood deployment where every node can hear at least 2 - 3 others directly, a hop limit of 3 generates significant redundant retransmissions. Consider reducing to 2 if traceroutes show no path requires more than 2 relays.

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

Local-only repeater that should not propagate far

Important: setting `hop_limit` low on a repeater does **NOT** limit the packets it relays for others — it only limits how far the repeater's *own* messages (its NodeInfo, position, telemetry) travel. To limit how far transit traffic propagates through your infrastructure you must coordinate settings network-wide (see below) or use rebroadcast-mode / role settings.

If you deploy a repeater specifically to bridge two buildings on the same campus - not to reach the wider regional mesh - you can reduce the reach of the traffic *it itself originates* by setting `hop_limit = 2` on its own packets (its own NodeInfo, position, telemetry). This does **not** prevent it from forwarding transit packets that arrive with a remaining count of 3, so it does not constrain its repeater function — it only limits the blast radius of traffic the repeater itself generates.

Note: the `lora.ignore_incoming` array is a **blocklist** — node IDs listed there have their packets dropped on receive — not a whitelist of nodes to serve. It is also deprecated in the client apps. To ignore a specific node, long-press it in the app and mark it ignored rather than relying on this setting.

The Broadcast Storm Risk From High Hop Counts

The maximum hop_limit is 7 (it cannot be set higher). Values of 6 and 7 are valid settings, not a forbidden range — but the Meshtastic project strongly recommends leaving hop_limit at 3 unless you have a specific, verified reason to raise it, because unnecessarily high hop counts cause network problems. Here is why:

- High hop limits in dense networks multiply rebroadcasts and waste airtime. Meshtastic's managed flooding suppresses duplicate (already-heard) rebroadcasts, which bounds the

effect well below a naive geometric explosion — but it mitigates rather than eliminates the congestion. Keep `hop_limit` as low as your path requirements allow.

- LoRa is a half-duplex medium. Each retransmission blocks all other transmissions in range for the duration of the air-time. Airtime per packet is payload-dependent: a long packet on the Long Slow preset (SF12, 125 kHz, ~180 bps data rate) can occupy the channel for roughly 2 - 3 seconds. Combined with high hop counts in a dense mesh this can push channel utilisation very high and severely degrade the network.
- Meshtastic **does** have a back-off mechanism: it uses CSMA/CA (similar to WiFi), not Ethernet's CSMA/CD. All transmitters perform Channel Activity Detection (CAD) before transmitting, and if the channel is busy they wait. The amount of wait is randomly picked from a contention window whose size grows with channel utilisation, specifically to limit collisions during congestion.

If you feel you need a high hop limit, the better solution is usually to add more infrastructure repeaters to shorten the required path, rather than raising the hop counter.

Configuring `hop_limit` via CLI

```
# Set hop limit to 3 (default - recommended for most networks)
meshtastic --set lora.hop_limit 3

# Set hop limit to 4 (for large spread-out networks after traceroute verification)
meshtastic --set lora.hop_limit 4

# Set hop limit to 2 (for small dense networks or local-only repeaters)
meshtastic --set lora.hop_limit 2

# Verify the applied value
meshtastic --get lora.hop_limit
```

`hop_limit` Is a Per-Node Origination Setting

An important subtlety: `hop_limit` controls the initial value placed in packets that *this node originates*. It has no effect on packets that arrive from another node already carrying a hop count - those are forwarded as received (after decrement). Therefore:

- Setting `hop_limit = 2` on a relay node limits only that node's own originations.
 - A packet originating elsewhere with `hop_limit = 5` will still traverse your relay with a remaining count of 4 after decrement - your local setting does not cap it.
 - To limit how far foreign traffic propagates through your infrastructure, you must coordinate `hop_limit` settings across all nodes in your deployment, or use channel-level policies.
-

Monitoring Channel Utilisation After Changes

After adjusting `hop_limit`, monitor channel utilisation via the [Meshtastic app](#) or web UI, where it is clearly labelled in the node detail view. Channel utilisation (ChUtil) is measured over a 1-minute window (six 10-second periods). Use these health bands consistently: green below 25%, orange 25 - 50%, red above 50% ChUtil; AirUtilTX should be much lower (a few percent). Note that ChUtil is a lagging rolling average, so it can read healthy while the channel is momentarily saturated. If you see utilisation spike after a `hop_limit` increase, your network density may not support the change and you should revert.

Fixed Position for Repeater Nodes

A repeater node that knows its own location serves the community in two ways: it appears accurately on coverage maps, and it lets neighbouring nodes calibrate their own position estimates. Without a fixed position, a GPS-less repeater either appears at coordinate (0, 0) in the ocean or does not appear on the map at all. This page explains how to configure a precise static position, how to reduce position precision for privacy, and how to minimise the air-time cost of periodic position broadcasts.

Why a Fixed Position Matters for Infrastructure Nodes

- **Coverage visibility** - Community members and emergency coordinators use the network map to understand what areas are served. A repeater with an accurate position lets them trace coverage boundaries and identify gaps.
 - **Traceroute path correlation** - When operators run traceroutes, hop positions appear on the map. A correctly placed repeater gives a geographically meaningful path diagram.
 - **SNR context** - Signal-to-noise ratio reports are much more useful when the receiving node's coordinates are known, because operators can correlate signal strength with distance and terrain.
 - **No GPS module required** - Many dedicated repeater platforms (e.g. RAK WisBlock with no GPS module, T-Beam with GPS disabled) can broadcast a fixed configured position without incurring GPS power draw.
-

Configuring a Static GPS Position Without a GPS Module

Method 1: Meshtastic CLI (recommended)

The `--setlat`, `--setlon`, and `--setalt` flags write a fixed position directly to the device's configuration storage. Once set, this position is broadcast as the node's location even with no GPS hardware present. This is the documented, recommended way to set a fixed position.

```
# Set position for a repeater at the top of Mount Davidson, San Francisco
# Latitude: 37.7406, Longitude: -122.4538, Altitude: 282 metres
meshtastic --setlat 37.7406 --setlon -122.4538 --setalt 282
```

Determine the coordinates of your site using Google Maps, CalTopo, or any mapping tool that provides WGS84 decimal degrees (the standard Meshtastic expects). Right-click the exact antenna location on Google Maps to copy coordinates.

Method 2: Meshtastic Python API

If you prefer scripting, the Python API exposes `setFixedPosition()` on the local node, which takes decimal-degree latitude and longitude plus an integer altitude in metres:

```
from meshtastic.serial_interface import SerialInterface

iface = SerialInterface()

# Set a fixed position: latitude, longitude (decimal degrees), altitude (metres)
iface.localNode.setFixedPosition(37.7406, -122.4538, 282)

iface.close()
```

Method 3: Meshtastic Web UI

Connect to the node's web interface (available on ESP32-based devices at `http://meshtastic.local` or the device's IP when connected to Wi-Fi). Navigate to **Config → Position**, enable *Fixed Position*, and enter latitude, longitude, and altitude. Save and reboot.

Position Precision: Reducing Exact Location for Privacy

If the repeater is on private property and the owner does not want the exact address broadcast on the mesh, reduce position precision. Position precision is a **per-channel** setting (`module_settings.position_precision`), so it is configured against a specific channel index rather than globally. Meshtastic supports precision levels from full resolution (sub-metre) down to a heavily-rounded approximation. The exact bits-to-radius mapping is defined by the in-app precision slider; the values below are approximate and may shift between firmware versions - confirm against the current Meshtastic position-config docs.

Common choices for community repeaters:

- **Full precision** (default) - broadcast exact coordinates. Acceptable for tower sites, mountain tops, and public-land installations.
- **~1.5 km precision** - rounds to roughly a 1.5 km radius. Identifies the general neighbourhood without revealing the specific building.
- **~12 km precision** - rounds to roughly a 11.7 km radius. Useful for regional-scale maps where sub-kilometre accuracy is unnecessary and the operator prioritises privacy.

```
# Position precision is per-channel; --ch-index selects the channel (0 = primary)
# Set position precision bits to 14 (approximately a 1.5 km radius)
meshtastic --ch-index 0 --ch-set module_settings.position_precision 14

# Set position precision bits to 11 (approximately a 11.7 km radius)
meshtastic --ch-index 0 --ch-set module_settings.position_precision 11

# Restore full precision (precision bits 32)
meshtastic --ch-index 0 --ch-set module_settings.position_precision 32
```

The `position_precision` value is a bit-field width - higher values mean more significant bits retained from the raw coordinate, i.e. higher accuracy. Values below about 10 round so aggressively that the position becomes almost meaningless for map purposes. Because this is a per-channel setting, you can carry different precision on different channels.

Position Broadcast Interval for Fixed Nodes

A fixed repeater does not move. By default the node broadcasts its position on a periodic interval (the firmware default is 15 minutes), and smart position broadcast - controlled by `position.position_broadcast_smart_enabled` (default `true`) - further reduces transmissions when the node is stationary. For a static infrastructure node you can simply lengthen the interval to at least 12 hours (43200 seconds), or even 24 hours (86400 seconds), to avoid wasting air-time that could be used by mobile nodes.

```
# Set position broadcast interval to 12 hours (43200 seconds)
meshtastic --set position.position_broadcast_secs 43200

# Set position broadcast interval to 24 hours (86400 seconds) - recommended for fixed sites
meshtastic --set position.position_broadcast_secs 86400

# Smart position broadcast is on by default; set false only if you want to disable it
meshtastic --set position.position_broadcast_smart_enabled false
```

With an 86400-second interval, a fixed node broadcasts its position approximately once per day (plus once at boot). The firmware default interval is 15 minutes; increasing it to the 12-24 h range for a static node saves many transmissions per day across a network with multiple infrastructure nodes.

Verifying the Fixed Position Was Applied

```
# Check position configuration on the device
meshtastic --get position

# Pull the device info including last known position
meshtastic --info | grep -A 5 position
```

After the node reboots following a position update, connect to the [Meshtastic app](#) and look at the node list. The repeater should appear at the correct location on the map within one position

broadcast interval.

Altitude Accuracy

Altitude in Meshtastic is stored in whole metres. It is commonly referenced to the WGS84 ellipsoid rather than mean sea level, but the `Position` message also carries an `ALTITUDE_MSL` flag that indicates when a value is given relative to mean sea level - so do not assume the value is always ellipsoidal. For antenna-height accuracy, use the elevation of the antenna itself, not ground level at the base of the tower. For most community mapping purposes, ground-level elevation from a topo map is sufficient - the difference rarely affects coverage visualisations.

Find accurate elevation using the USGS National Map (`apps.nationalmap.gov`) or `open-elevation.com/api/v1/lookup?locations=LAT,LON` for non-US sites.