MeshCore Sensor Nodes

MeshCore Sensor Support

MeshCore does **not** ship a separate "Sensor" firmware variant. MeshCore device firmware is built around three roles - **Companion**, **Repeater**, and **Room Server**. Environmental-sensor support is a **compile-time feature**: it is enabled when the firmware is built with the relevant `ENV_INCLUDE_*` flags (the environment-sensor manager, demonstrated by example builds such as `simple_sensor` / `SensorMesh`). It is an evolving, limited capability, not a polished deployable role you "flash and go."

Because sensor support is selected at build time, there is no stock "RAK4631 SENSOR" release asset to download. To use it you compile a sensor-capable build for your board. nRF52 boards (RAK4631, T-Echo) are flashed via UF2 drag-and-drop or the web flasher; `esptool` applies only to ESP32 boards.

Supported Sensors

When a sensor-capable build is compiled in, the environment-sensor manager probes the I2C bus to detect supported sensors. Commonly referenced hardware includes:

- **BME280** - temperature, humidity, barometric pressure (I2C). A common choice for weather monitoring.
- **BME680** - adds a gas/VOC sensor to the BME280 feature set, useful for indoor air-quality indication. (IAQ requires the Bosch BSEC library and a calibration period to be meaningful.)
- **INA219** - bus voltage and current measurement over I2C. Useful for monitoring battery banks, solar panels, or load/charge current.
- **Custom sensors** - additional inputs require *modifying and recompiling the firmware* (e.g. the `onSensorDataRead` hook plus the appropriate `ENV_INCLUDE_*` flags in `EnvironmentSensorManager`). This is a build-time change, not a runtime configuration option.

How Sensor Telemetry Is Delivered

A common misconception is that MeshCore "broadcasts" sensor readings in its adverts. It does not. A MeshCore **advert** announces a node's presence and routing information only - its name, optional

position, and signed public key. Adverts do **not** carry temperature, humidity, or any other sensor value.

Sensor data travels instead over MeshCore's **binary request/response protocol**. A client explicitly requests telemetry from a node (`REQ_TYPE_GET_TELEMETRY_DATA`); the node replies with a **CayenneLPP**-encoded payload. This exchange is addressed and permission-gated - it is a pull (request/response), not a broadcast that every node receives and forwards. (Adverts themselves are either zero-hop or flooded for routing purposes, but in neither case do they contain sensor readings.)

Power Profile

A telemetry-reporting MeshCore node can run at a low duty cycle, which makes solar or long-term battery deployment feasible. Concrete figures depend heavily on TX power, reporting frequency, whether Bluetooth is disabled, and the board's sleep current, so the numbers below are **rough estimates, not specifications**:

- An nRF52840 node (e.g. RAK4631) with a BME280, reporting infrequently with BLE off and low TX power, can draw on the order of a few mAh/day. Measure your own node's consumption before sizing a battery - do not treat any single figure as a firm spec.
- Battery runtime estimates must also account for self-discharge and capacity derating over time, so a nominal pack capacity does not translate directly into a guaranteed number of days.
- A small (~1 W) solar panel can offset consumption at a site that *reliably* gets 4+ peak-sun-hours, but this varies by season and location and will not keep a node running "indefinitely" - lithium cells still age out over calendar years.
- For the lowest power, disable Bluetooth and set LoRa TX power to the minimum needed to reach the nearest repeater.

Outdoor lithium safety: never charge a lithium cell (including LiFePO4) below 0 C / 32 F. For any outdoor node deployed where winter temperatures drop below freezing, use a charger/controller with a low-temperature charge cutoff, and add an in-line fuse on the battery positive lead.

Configuration

MeshCore is configured over USB/serial or over BLE (via the companion app or the MeshCore CLI). The actual sensor CLI is a small generic key/value store - there is **no** `set sensor type`, `set sensor addr`, `set sensor interval`, or `set sensor enabled` command. The real commands are:

- `sensor list [start]` - list the sensor custom variables (printed as `key=value` lines).
- `sensor get <key>` - read a sensor custom variable.
- `sensor set <key> <value>` - set a sensor custom variable.

Which sensors are present is determined at **compile time** (the `ENV_INCLUDE_*` build flags) and by I2C auto-detection at startup - not by a runtime enable/disable or type-selection command. I2C addresses are detected during the environment sensor manager's bus probe; they are not set from the CLI. Advert cadence is governed by the real interval prefs (e.g. `set flood.advert.interval <hours>` / `set advert.interval <minutes>`), which control advert timing - they do not broadcast sensor readings.

Use Cases

- **Remote weather station:** BME280 on a ridge or mountain peak, reporting temperature, humidity, and pressure on request.
- **Water level monitoring:** an ultrasonic or pressure sensor at a stream crossing or water tank. Treat this as **best-effort situational awareness only** - mesh packets can drop, and a hobbyist sensor must *not* replace official NWS/USGS flood warnings for any safety decision (see the Water Quality and Flood Monitoring page).
- **Air quality indication:** BME680 in an urban or wildfire smoke corridor (low-cost sensors are indicative, not reference-grade).
- **Soil temperature for agriculture:** BME280 buried at root depth in a remote field.
- **Power system monitoring:** INA219 across the shunt resistor of a solar-charged battery bank, reporting **bus voltage and load/charge current**. Note: state of charge is not measured directly - it must be derived (e.g. coulomb counting) from voltage and current.

Integration with Data Pipelines

Capturing MeshCore sensor data is not automatic - readings do not appear by themselves in the app's node list. You need a node connected to a host (a room server, or a serial/USB/BLE bridge) running code that requests telemetry and logs it. Use the real async **meshcore_py** library: create a client, subscribe to telemetry events, and issue telemetry requests through `commands.*` - do not parse adverts for sensor values.

```
import asyncio, sqlite3, datetime
from meshcore import MeshCore, EventType

async def main():
    # create_serial(port), create_tcp(host, port), or create_ble(address)
    mc = await MeshCore.create_serial("/dev/ttyUSB0")

    db = sqlite3.connect("sensors.db")
    db.execute("CREATE TABLE IF NOT EXISTS readings "
               "(ts TEXT, node TEXT, payload TEXT)")
```

```
    def on_telemetry(event):
        # event.payload is the decoded CayenneLPP telemetry response
        db.execute("INSERT INTO readings VALUES (?,?,?)", (
            datetime.datetime.utcnow().isoformat(),
            str(event.payload.get("from", "")),
            str(event.payload),
        ))
        db.commit()

    mc.subscribe(EventType.TELEMETRY_RESPONSE, on_telemetry)

    # Pull telemetry from a known contact (request/response, not broadcast):
    contacts = await mc.commands.get_contacts()
    for contact in contacts.values():
        await mc.commands.req_telemetry(contact)

    # keep the asyncio loop running to receive responses
    await asyncio.Event().wait()

asyncio.run(main())
```

There is no `MeshCore("/dev/ttyUSB0")` constructor, no `mc.on_advertisement = ...` callback, no `mc.run()`, and no `advert.get("temperature")` - adverts do not carry sensor values. MeshCore has **no built-in MQTT module**; MQTT export is done through an external bridge (meshcore_py-based, or a third-party tool such as MeshMonitor), not by the room server.

A MeshCore Home Assistant integration does exist (github.com/meshcore-dev/meshcore-ha, domain `meshcore`), built on meshcore_py.

Comparison with Meshtastic Telemetry

Feature	Meshtastic Telemetry module	MeshCore sensor support
BME280 / BME680 support	Yes (auto-detected on I2C)	Yes, via built-in environmental telemetry (compiled in)
INA219 support	Yes (via power metrics)	Yes

Feature	Meshtastic Telemetry module	MeshCore sensor support
Data propagation	Mesh packet (Telemetry protobuf)	CayenneLPP telemetry over the binary request/response protocol (pull, not broadcast)
MQTT output	Yes (via Meshtastic MQTT module, ESP32 gateways)	No built-in MQTT; requires an external bridge (e.g. meshcore_py or MeshMonitor)
Dedicated sensor firmware	No (runs on standard node firmware)	No separate variant - sensor support is compiled into the node firmware via `ENV_INCLUDE_*` flags
Power profile	Low - dominated by TX duty cycle and sleep config	Low - likewise dominated by TX duty cycle and sleep config

---

Revision #3
Created 2026-05-03 05:29:11 UTC by Mesh America Admin
Updated 2026-06-10 03:08:48 UTC by Mesh America Admin