Skalman on Home Assistant

This is the story of how we got our robot lawn mower — a Husqvarna Automower 310 that goes by the name **Skalman** — to show up in Home Assistant, complete with battery status, what it's doing right now, and buttons to start, pause and send it home. It sounded simple in theory. It wasn't.

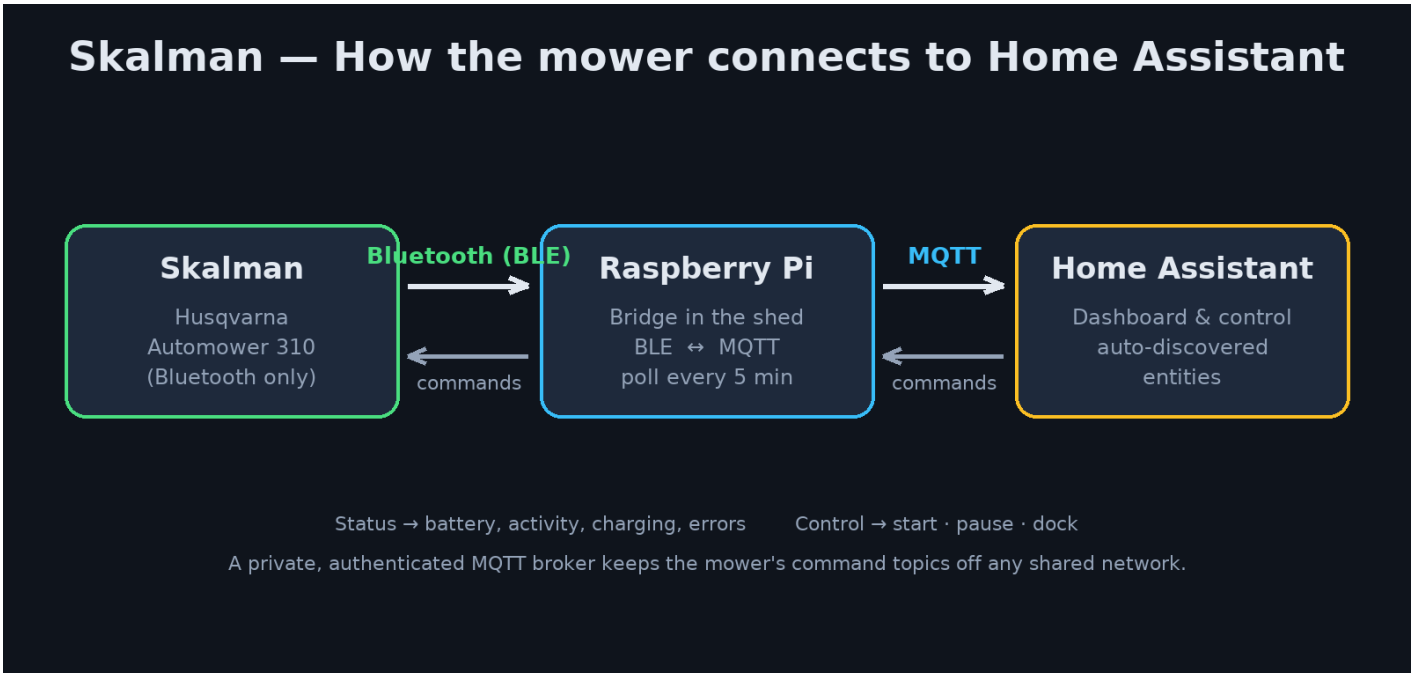
Background

Skalman is a Husqvarna Automower 310 from before the “Mark II” refresh — a design Husqvarna kept selling new well into the early 2020s, even though under the hood it stayed decidedly old-school (more on that below). There's no cloud service we can reach — the "Automower Connect" menu on the mower is locked behind a "contact your dealer" message, so the normal route via Husqvarna's app and official integration was closed to us. The only thing we had to work with was **Bluetooth** (BLE), which the mower speaks locally.

The problem with Bluetooth is range. The mower and its charging station sit a fair distance away, and our Home Assistant server can't reach that far over Bluetooth. So we needed something close by.

How the solution works

The answer turned out to be a small **Raspberry Pi placed out near the mower**. It acts as a bridge: it talks Bluetooth to Skalman on one side, and passes the information on to Home Assistant over a messaging protocol called MQTT on the other side.



In broad strokes: the Pi sits close enough to get a decent Bluetooth link, and it's running around the clock anyway since it looks after other things out in the shed. Home Assistant never has to care about Bluetooth itself — it just subscribes to what the bridge publishes.

The Bluetooth pairing — the tricky bit

This was the part that took by far the longest, and it's worth writing down so we don't have to sweat as much next time.

A modern Bluetooth stack (like the one in a recent Raspberry Pi) wants to pair using the most secure method by default: encrypted "secure connections" with protection against eavesdropping. This mower can't do that. Despite being bought new in the 2020s, the pre-“Mark II” Automower 310 is an older design under the hood — it only speaks the simpler, older dialect ("legacy pairing" using the "Just Works" method). The result is that the mower politely replies "pairing not supported" and hangs up — and you sit there scratching your head.

The key was to **force the Bluetooth controller on the Pi to speak the old dialect** and switch off the demand for the advanced security mode. Once both sides were speaking the same language, it went through.

One extra hurdle: because the pairing menu on the mower is locked, you can't put it into pairing mode the usual way. The trick is to **flip the main power switch off and on** — that opens a three-minute pairing window on the mower. During that window the phone's Bluetooth needs to be off (the mower only lets one device in at a time), and then you run the pairing on repeat until it takes. For us it took a couple of attempts.

One important detail: the Bluetooth pairing itself and the mower's PIN code are **two completely separate layers**. The Bluetooth bond needs no code, whereas the machine's PIN is sent by the software afterwards, at the application level. Good to know when troubleshooting — you can have a working Bluetooth link and still be rejected because the PIN layer is misbehaving, or the other way around.

> **Note:** The Bluetooth bond is tied to the specific Bluetooth adapter. If we swap the dongle, or move to a different device down the line, we'll have to run the whole pairing procedure again.

The bridge that ties it all together

The Pi runs two parts:

- **A reader** that does exactly one thing at a time: connects to Skalman, reads the status (or sends a command), and disconnects cleanly again. It uses a clever library that reuses a cached device reference instead of scanning the airwaves every time, which makes the connection considerably faster. A lock makes sure only one Bluetooth session runs at a time, so nothing collides.
- **The bridge itself**, which runs the reader at regular intervals (every five minutes) and publishes the result to MQTT. It also sends out so-called *auto-discovery*, which means Home Assistant automatically creates the right entities without us having to configure anything by hand. In the other direction, the bridge listens for commands from Home Assistant and translates them into the mower's language.

Everything runs as a service that starts automatically and restarts itself if something goes wrong.

What showed up in Home Assistant

Once discovery was in place, a row of entities popped up, entirely automatically:

- Battery level
- What the mower is doing right now (mowing, heading out, heading home, charging, parked …)
- Charging status
- Any error codes
- And a proper **lawn mower control** with buttons to start, pause and dock

The mower's many different modes are translated into a handful of understandable states in the interface, so the card in Home Assistant simply shows "mowing", "docked", "paused" or "error".

Security

To begin with, the message traffic lived on a **shared community broker**. We moved away from that fairly quickly and shifted everything to our **own, private broker with a login**. The reasoning is simple: the command topics are the way in to actually *control* the mower. If they sit on a broker that others can reach, others could in theory send the mower off. Now the traffic is private and authenticated.

A bug we had to chase

A fun detail worth remembering: for several days Skalman reported nice status in Home Assistant, but the **Play button did nothing** — the mower refused to leave the dock.

It turned out the start command was being translated into "resume", which only works if the mower is already in the middle of a paused mowing session. From a docked/parked/schedule-controlled state, nothing happened at all. The fix was to make Play send a "mow now" command instead, one that bypasses the schedule and sends it out for a while straight away. After that it obeyed.

Range and reliability

Bluetooth through a shed wall isn't magic. The signal is fairly weak, and it shows:

- **When Skalman is on the dock**, the link is stable and the readings work fine.
- **When it's out mowing** and moving around the garden, the bridge often misses a reading. That's no big deal — the bridge logs it, skips it, and tries again next cycle without crashing.

If we want rock-solid status even while mowing, there are a couple of ways forward: hold a Bluetooth connection open the whole time instead of reconnecting over and over, or put a Bluetooth repeater closer to the garden. But for our needs — knowing the battery and mode, and being able to control it — today's solution does the job nicely.

Bonus: a rain guard

Since we had control over the mower anyway, we built a little rain guard. We don't have our own rain sensor, so we use the weather forecast. When it's raining, Skalman is kept on the dock, and it also has to wait an adjustable while after the rain stops so the grass can dry out a bit before heading out again. There are controls in the interface to switch the protection on and off and to adjust how long it should wait. The rain guard respects the mower's own schedule — it only releases the hold, it never forces it out.

Setting this up yourself

If you've got a Bluetooth-era Husqvarna Automower (the pre-"Mark II" models that only talk BLE locally, with no reachable cloud) and you'd like the same thing, here's the whole recipe. None of it is specific to our setup — plug in your own mower and broker details.

What you'll need

- **A Bluetooth Automower** without working "Automower Connect" cloud. If yours *does* have the cloud, use Husqvarna's official integration instead — this guide is for the ones that don't.
- **A small always-on Linux machine near the mower**, with Bluetooth — a Raspberry Pi 4/5 (built-in BT) or any Linux box with a USB BLE dongle. "Near" matters: BLE through a wall is weak, so the closer to the dock, the better.
- **Home Assistant**, already running, with the MQTT integration.
- **An MQTT broker** (Mosquitto is the usual pick). Run it locally and give it a username and password — you do *not* want the mower's command topics sitting on a broker just anyone can reach.
- **Your mower's PIN** (the code you use in the app / on the mower).
- The open-source `husqvarna-automower-ble` Python library (on PyPI / GitHub), which does the actual Bluetooth talking. It's the same library Home Assistant's own Husqvarna BLE support is built on.

Step 1 — Pair with the mower over Bluetooth

This is the fiddly part, so take it slowly. Modern Linux Bluetooth defaults to the most secure pairing method; the older mowers can't do that and will just refuse. You have to tell the Bluetooth controller to fall back to the old "legacy pairing / Just Works" method first.

On the Pi, as root:

```
btmgmt power off
btmgmt sc off        # secure connections OFF -> legacy pairing
btmgmt bondable on
btmgmt io-cap 3      # NoInputNoOutput -> "Just Works", no PIN prompt
btmgmt power on
```

Then, in the **same** `bluetoothctl` session (this matters — registering the agent elsewhere lets the secure method sneak back in):

```
agent NoInputNoOutput
default-agent
pair AA:BB:CC:DD:EE:FF     # your mower's BLE address
```

Two things to know:

- **Getting the mower into pairing mode:** if its Bluetooth menu is locked (many are), just flip the main power switch off, wait for the lights to go out, and back on. That opens a roughly three-minute pairing window. Your phone's Bluetooth needs to be off during it — the mower only bonds one device at a time.
- **Run the `pair` command on repeat** inside that window; it often takes a couple of tries to catch. Once you see `Paired: yes`, you're bonded for good and it reconnects on its own after that.

Good to know: the bond is tied to that specific Bluetooth adapter. Swap the dongle later and you'll pair again. (The mower's PIN is a separate layer from the Bluetooth bond — the bond needs no code; the PIN is sent by the library afterwards, at the app level.)

Step 2 — A reader that talks to the mower

Write one small script whose only job is: connect, read the status (and optionally send a command), disconnect cleanly. Keep it to a single Bluetooth session at a time — a lock file stops a scheduled poll and a manual run from colliding.

The core, trimmed right down:

```
from husqvarna_automower_ble.mower import Mower
from husqvarna_automower_ble.protocol import ResponseResult
from bleak_retry_connector import get_device

ADDRESS    = "AA:BB:CC:DD:EE:FF"   # your mower
CHANNEL_ID = 1197489078            # any fixed 32-bit number, kept constant
PIN        = 1234                  # your mower's PIN
```

```
mower  = Mower(CHANNEL_ID, ADDRESS, PIN)
device = await get_device(ADDRESS)          # fast: uses the bonded/cached device
if await mower.connect(device) == ResponseResult.OK:
    battery  = await mower.battery_level()
    charging = await mower.is_charging()
    state    = await mower.mower_state()
    activity = await mower.mower_activity()
    # commands: mower.mower_park() / mower.mower_pause() / mower.mower_override()
    await mower.disconnect()
```

Two hard-won tips:

- Feed `connect()` a **cached device** from `get_device()` rather than a fresh 30-second scan — much faster, and one less thing to fail.
- The mower releases its (single) Bluetooth link slowly. Reconnect too soon and you'll get `NOT_ALLOWED`, so use a generous retry backoff (~20s) and a cooldown between sessions.

Step 3 — A bridge to MQTT (with auto-discovery)

The bridge runs forever: every few minutes it runs the reader, publishes the result to MQTT, and it listens for commands coming back from Home Assistant. The nice trick is **MQTT discovery** — publish one small retained config message per entity and Home Assistant creates them for you, no YAML.

Publish state as JSON to a topic like `mower/state` (retained), plus `mower/availability` (`online` / `offline`). Then publish discovery configs such as:

```
# a battery sensor
"homeassistant/sensor/mower_battery/config": {
    "name": "Battery", "unique_id": "mower_battery",
    "state_topic": "mower/state",
    "value_template": "{{ value_json.battery }}",
    "device_class": "battery", "unit_of_measurement": "%",
}

# the lawn-mower entity itself
"homeassistant/lawn_mower/mower/config": {
    "unique_id": "mower",
```

```
    "activity_state_topic": "mower/state",
    "activity_value_template": "<template mapping raw state -> mowing/docked/paused/error>",
    "dock_command_topic":        "mower/cmd/dock",
    "pause_command_topic":       "mower/cmd/pause",
    "start_mowing_command_topic": "mower/cmd/start",
}
```

Map the mower's many raw states onto Home Assistant's four lawn-mower activities with a template — e.g. `CHARGING` / `PARKED` / `RESTRICTED` → `docked`, `MOWING` / `GOING_OUT` / `GOING_HOME` → `mowing`, `PAUSED` / `STOPPED` → `paused`, anything error-ish → `error`.

Subscribe to the three command topics and map each to a library call:

```
{ "mower/cmd/dock":  "park",
  "mower/cmd/pause": "pause",
  "mower/cmd/start": "override" }   # <- note this one
```

The one gotcha that will bite you: map the Start button to **override** ("mow now"), not **resume** . `resume` only un-pauses an already-paused session — from the dock it does nothing, and you'll swear the Start button is broken. `override` tells the mower to go out and mow now, bypassing the schedule.

Finally, run the bridge as a `systemd` service with `Restart=on-failure` so it comes back after a reboot or a hiccup.

Step 4 — Point Home Assistant at the broker

In Home Assistant, set the MQTT integration to your broker's address and the username/password you created. That's it — within a poll cycle the mower's entities appear on their own (battery, state, activity, charging, and a lawn-mower card with Start / Pause / Dock). No manual entity setup.

Step 5 (optional) — a rain guard

As described above: a couple of helper toggles plus automations that watch your weather integration and keep the mower docked while it's raining, and for a set while after. Purely optional, but a nice touch.

The two things that cost us the most time were the legacy-Bluetooth pairing in Step 1 and the resume-vs-override mix-up in Step 3. Get those two right and the rest is just plumbing.

Status

Skalman is up and running: status shows in Home Assistant, and start, pause and dock all work. What's left is mostly polish — above all making the readings more reliable while it's out and moving. But the foundation is solid, and it's mowing away.

Revision #6
Created 2026-07-10 15:04:53 UTC by Admin
Updated 2026-07-10 15:38:44 UTC by Admin