ChatGPT

AutoPi API endpoints and metrics useful for a Home-Assistant integration

AutoPi exposes a REST API that allows third-party clients to query information about vehicles and devices. The API follows an OpenAPI schema and most calls use a `GET` method with JSON responses. The endpoints below are organised by the type of information they expose and include typical query parameters. These calls require an authenticated API token.

1. Vehicle position and movement

Metric/thing of interest	API call & parameters	Notes
Last known position of a specific device or vehicle	`GET /logbook/v2/most_recent_position/` – parameters: `device_id` (UUID) or `vehicle_id` (int). Returns a `RecentPosition` object with timestamp, UTC time, *course_over_ground* (heading in degrees), *speed_over_ground*, altitude, number of satellites (`nsat`) and a `location` object containing latitude and longitude [1].	Use this to feed a Home-Assistant `device_tracker` entity. It provides both GPS position and immediate motion data (speed/heading/altitude).
Last known position for all devices	`GET /logbook/v2/most_recent_positions/` – returns an array of `RecentPositions` objects for each unit [1]. Each entry includes the timestamp and an array of positions for the device.	Useful for fleet dashboards or if multiple vehicles are being tracked.
Last known position of a vehicle (legacy)	`GET /logbook/most_recent_vehicle_positions/` – returns the most recent positions for all devices in the customer account [2]. Response schema is the same `RecentPositions`.	Equivalent to the v2 call but labelled "vehicle" – either call can be used.
List of trips (start/end times and coordinates)	`GET /logbook/v2/trips/` – accepts optional query parameters to filter by start and end times or by device/vehicle. Returns a paginated list of `TripSerializerV2` objects [3]. Each trip includes `start_time_utc`, `end_time_utc`, `start_position_lat/lng`, `end_position_lat/lng`, distance in kilometres (`distanceKm`), duration and a tag (`business` / `personal`) [3].	A Home-Assistant integration could create sensors for the latest trip's start location, end location, distance and duration.
Details of a single trip	`GET /logbook/v2/trips/{id}/` – supply the trip UUID to retrieve one `TripSerializerV2` object [4].	Useful for retrieving details of a specific journey.

Metric/thing of interest	API call & parameters	Notes
Aggregated time & distance metrics	`GET /logbook/fleet_summary/timedistance/` – optional parameters: `vehicle_id` (list of ints), `device_id` (list of UUIDs) and `exclude_unassociated` (bool). The call returns a `TimeDistanceData` object with two sections: *today* and *month*. Each section contains `distance_vehicle`, `distance_fleet`, `time_vehicle`, `time_fleet`, `date`, `first_day_in_month` and `last_day_in_month` [5].	Provides high-level daily and monthly distance and driving time. Can be exposed as sensors representing total distance driven today or this month.
Trip summary (deprecated)	`GET /logbook/trips/` – similar to the v2 call but marked as deprecated [6].	Use the v2 version instead.
Number of vehicles on/ off location	`GET /logbook/fleet_summary/vehicles/` – optional `vehicle_id` and `exclude_unassociated`. Returns `VehiclesData` containing counts of all vehicles, active vehicles now, vehicles driven in the last 30 days, vehicles currently on location, off location, etc. [5].	Could be used to expose counts such as “vehicles on-site” or “vehicles off-site”.
Geofence summary	`GET /logbook/fleet_summary/geofences/` – optional `device_id`. Returns `GeofenceSummary` with number of geofences, number of devices using a geofence and a list of each geofence [7].	Useful if Home-Assistant needs to display geofence statistics or number of vehicles in/ out of zones.
Diagnostic codes summary	`GET /logbook/fleet_summary/diagnostics/` – optional `from_utc`, `end_utc` and `search`. Returns `DiagnosticsData` containing the total number of diagnostic trouble codes and a list of codes [8].	Allows Home-Assistant to expose sensors such as “number of active DTCs”.
Alerts summary	`GET /logbook/fleet_summary/alerts/` – optional filters such as severity and date range. Returns `AlertsData` with total alerts and a breakdown by severity [8].	Could be used to notify about unresolved alerts or high-severity events.

2. Battery and connection information

Metric/thing of interest	API call & parameters	Notes
Latest battery voltage and percentage	`GET /logbook/recent_stats/` – parameters: `device_id` (UUID) and `from_timestamp` (ISO date). Returns a `RecentStats` object with `voltage`, `voltage_ts` (timestamp of voltage reading), `voltage_level` (battery percentage) and `voltage_level_ts` [9]. It also includes `latest_com`, the time of last communication with the device [9].	Suitable for a `sensor` entity representing battery voltage and percentage. The last communication timestamp can be used to detect offline devices.
Most recently logged telemetry (multiple metrics)	`GET /logbook/v2/recent_stats/` – parameters: `device_id` (required), `from_timestamp` and optional `type` (e.g., `obd.bat`) and `kind` (`events` or `data`). Returns an array of `RecentStatsV2` items where each item has fields `@ts` (timestamp), `@rec` (record upload time), `@uid` (unit ID), `@vid` (vehicle ID) and `@t` (data type such as `obd.speed` or `obd.rpm`) [1]. Additional dynamic keys contain the actual values.	This endpoint is the main way to retrieve the *latest* sensor values. Without the `type` filter it returns the most recent value of every data type. Setting `type=obd.speed` will return the latest vehicle speed; `type=obd.rpm` returns engine RPM; `type=obd.fuel_level` returns fuel level, etc.
Available data fields and last value	`GET /logbook/storage/data_fields/` – parameters: `device_id` (UUID) or `vehicle_id` (int). Returns an array of `Field` objects, each containing `field_prefix`, `field_name`, sampling `frequency`, `type`, `title`, `last_seen` and `last_value` [10].	Use this call to discover all telemetry fields supported by a device – it includes OBD PIDs, battery metrics, GPS values and custom sensors. The `last_value` element provides the most recent reading so sensors can be created dynamically.
Battery diagnostics and other OBD data via manual query	`POST /dongle/{unit_id}/obd/query/{command}/` – supply the unit ID and a standard OBD command string (e.g., `010C` for RPM or `012F` for fuel level). Returns the result of that command [11].	Use this when a particular PID is not logged automatically. The call is synchronous; results may take some time.
Diagnostic trouble codes (DTCs)	`GET /dongle/{unit_id}/obd/dtc/` – returns the current diagnostic trouble codes for the unit [11]. `POST /dongle/{unit_id}/obd/dtc/clear/` clears them [11].	You could expose a binary sensor indicating whether DTCs are present and provide a service to clear codes.

3. Raw sensor data and historical telemetry

Metric/thing of interest	API call & parameters	Notes
Raw historical telemetry (time series)	`GET /logbook/raw/` or `GET /logbook/storage/raw/` – parameters: `device_id` (UUID), `start_utc`, `end_utc`, optional `data_type` (e.g., `obd.speed`), `page_num` and `page_size` [9]. The response is a `PagedRawData` object containing `RawData` items; each item includes the recording timestamp (`ts`), upload time (`rec`) and `t` (data type) [9].	Use this endpoint to retrieve historical time-series for graphs. Without `data_type` it will return all logged data. Each result contains the value in a dynamic field named after the data type.
Aggregated averages and min/max over intervals	`GET /logbook/storage/read/` – parameters: `device_id`, `field` (data field name), `field_type` (prefix), `from_utc`, optional `to_utc`, `interval` (e.g., `10m`) and `aggregation` (`avg`, `min`, `max`) [12]. Returns a `DataItem` list with aggregated values over specified intervals.	Ideal for Home-Assistant's statistics sensors or energy dashboards. For instance, to compute average speed every hour set `interval=1h` and `aggregation=avg`.
Simplified events	`GET /logbook/simplified_events/` – parameters include `device_id`, `vehicle_id`, `start_utc`, `end_utc`, optional `type` and `tags` [13]. Returns an array of paged events; each event has timestamp `ts`, `event` type, `tag`, `name` and `kwargs` (additional data).	Exposes events such as harsh braking, speeding alerts, geofence entry/exit or custom triggers. Can be converted into Home-Assistant events or notifications.

4. Useful aggregated dashboards

Metric/thing of interest	API call & parameters	Notes
Device summary (active devices, updates, alerts)	`GET /logbook/fleet_summary/devices/` – optional `device_id` and `alert_type`. Returns `DevicesData` with counts of all devices, active devices now and in the last day/ month, the latest firmware version and dictionaries of active alerts and update status [8].	Use this to create a dashboard showing how many AutoPi units are online or need updates.
Vehicle summary (driven/not driven, on/off location)	`GET /logbook/fleet_summary/vehicles/` – returns `VehiclesData` with counts of all vehicles, active vehicles now, vehicles driven in the last 30 days, not driven vehicles, vehicles on location and off location [5].	Could be represented as sensors for "number of vehicles driven this month" or "vehicles on premises".

Metric/thing of interest	API call & parameters	Notes
Geofence summary	`GET /logbook/fleet_summary/geofences/` – returns counts of geofences and devices using them, plus a list of geofence details [7].	Useful for integration with Home-Assistant's zone system.
Diagnostics summary	`GET /logbook/fleet_summary/diagnostics/` – returns total number of diagnostic codes and a list of codes [8].	Allows a Home-Assistant sensor to indicate the number of active DTCs.
Alerts summary	`GET /logbook/fleet_summary/alerts/` – returns the total number of alerts and a breakdown by severity [8].	Could be used to show "open alerts" in a dashboard.

5. Discovering and using OBD sensor types

The API does not list every available OBD parameter explicitly, but the `type` or `data_type` query parameters on the **recent stats** and **raw** endpoints accept strings identifying the measurement. Some common values include:

- `obd.speed` – vehicle speed.
- `obd.rpm` – engine revolutions per minute.
- `obd.coolant_temp` – coolant temperature.
- `obd.fuel_rate` – fuel consumption rate.
- `obd.throttle_pos` – throttle position percentage.
- `obd.engine_load` – engine load percentage.
- `obd.mass_air_flow` – mass air flow sensor.
- `obd.intake_temp` – intake air temperature.
- `obd.fuel_level` – fuel tank level percentage.
- `obd.bat` – battery voltage (also available via the legacy `/logbook/recent_stats/` endpoint).

A Home-Assistant integration should call `/logbook/storage/data_fields/` [10] during setup to determine which of these fields are logged for a specific device. Each `Field` object in the response includes the `field_prefix` (e.g., `obd`) and `field_name` (e.g., `speed`), allowing the integration to build a list of sensors automatically.

Summary

The AutoPi API exposes rich telemetry for vehicles. For a Home-Assistant integration you should start by obtaining an API token, then use `GET /logbook/v2/most_recent_position/` and `/logbook/v2/most_recent_positions/` to track vehicle location and movement. Use `GET /logbook/v2/recent_stats/` to poll the latest readings of speed, RPM, fuel level or any other OBD sensor; it returns a value per data type. For historical charts or statistics, use `GET /logbook/raw/` or `GET /logbook/storage/raw/` and optionally `GET /logbook/storage/read/` for aggregated averages. Aggregated summaries of distance/time, devices, vehicles, geofences and diagnostics are available through the fleet-summary endpoints. Together, these calls provide a comprehensive set of metrics for building detailed Home-Assistant dashboards or automations.

1 2 3 4 5 6 7 8 9 10 11 12 13 api.autopi.io

https://api.autopi.io/