AURA2 · DEVELOPER DOCUMENTATION

Communication and API

How your software reads the Aura2 over USB and Wi-Fi: commands, JSON answers, HTTP endpoints, the environmental log and working examples.

Qualimetric · Nijmegen, The Netherlands · info@qualimetric.nl · nexchrom.nl · 6 November 2026

CONTENTS

1. Overview
2. USB
3. Commands
4. Measurements
5. Device state
6. Wi-Fi and the HTTP interface
7. The environmental log
8. Settings
9. Versions and compatibility
10. Example: reading the Aura2 from Python

Version **1.1** (November 2026) · applies to protocol version **21**, firmware **0.22.0** and HTTP API **v1**. The Aura2 is in development; this document is updated along with it.

This document describes how software talks to the Aura2: over **USB** (a serial line protocol) and over **Wi-Fi** (a read-only HTTP/JSON interface). It is written for developers who want to read the Aura2 from their own software. For installation, placement and day-to-day use, see the instruction manual.

1. Overview

The Aura2 offers the same measurement data on two channels.

	USB-C	Wi-Fi
Role	primary channel	second, optional channel
Transport	USB serial port (CDC), one JSON line per answer	HTTP on port 80, path `/api/v1/`
Read measurements, status, log files	yes	yes
Change settings, clock, Wi-Fi, delete log files	yes	**no**
Integrity check	CRC-32 on every answer	by TCP
Default	always available	**off** until switched on over USB

Three principles hold throughout:

- **Measurements requested by your software carry no timestamp.** Your software timestamps them with its own clock at the moment of the request, so one clock governs your records.
- **The device refuses what it cannot do correctly.** Unknown commands, unknown fields and settings outside their limits are rejected with a reason, never silently ignored or adjusted.
- **Nothing can be changed over Wi-Fi.** Settings, the clock and the Wi-Fi configuration itself can only be changed over USB.

Security

The Wi-Fi interface has **no password and no encryption**. Anyone on the same network can read the measurements, the status and the log files. Nobody can change anything over Wi-Fi: the worst that can happen is that someone reads along. We recommend connecting the Aura2 to a **separate, shielded laboratory network**, and leaving Wi-Fi switched off if you do not use it.

---

2. USB

2.1 Connection

The Aura2 appears as a serial (COM) port; on Windows 10/11 no driver is needed.

Setting	Value
Baud rate	115200 (nominal; the USB connection ignores it)
Data format	8 data bits, no parity, 1 stop bit
DTR	**must be active**. The device only sends once the host has set DTR. Most serial libraries do this by default
RTS	inactive
Character set	UTF-8 (in practice ASCII)

Identify an Aura2 by its answer to `ID`, not by its USB vendor or product ID. To find the device, open each candidate port, send `ID` with a short timeout (2 s is ample) and check that `dev` is `"aura2"`.

Only **one program at a time** can have the port open.

2.2 Commands and answers

Your software sends a command as plain text, terminated by a newline (`\n`; `\r\n` is also fine). The Aura2 replies with **exactly one line**:

```
<json>|<crc32>
```

- `<json>` is a single JSON object without line breaks.
- `<crc32>` is 8 lower-case hexadecimal digits: the CRC-32 (IEEE 802.3, the same as `zlib.crc32` in Python) over the bytes of `<json>`, that is, everything before the last `|`. Check value: CRC-32 of `123456789` is `cbf43926`.

Check the CRC on every answer and discard the answer if it does not match. Split on the **last** `|`, because the JSON may itself contain that character.

Rules for commands:

Rule	Detail
Case-insensitive	`id` works the same as `ID`
Leading and trailing whitespace is ignored	so a `\r` does no harm
An empty line gets no answer	do not wait for one
At most 767 characters per line	a longer line is rejected and discarded up to the newline
One command at a time	send the next command only after the answer to the previous one

Every answer contains `cmd` (the command it answers) and `ok` (`true` or `false`). **Check that `cmd` matches the command you sent**, so that an answer stream that has got out of step is noticed instead of delivering the wrong

values.

One rule to know when you check this: a **successful** `CONFIG=`, `TIME=` or `WIFI=` answers with the state as it is after the change, which is the answer of the matching read command. Its `cmd` is therefore `CONFIG?`, `TIME?` or `WIFI?`. A **failed** one carries the command you sent (`CONFIG=`, `TIME=`, `WIFI=`) with `ok: false`. So accept, for example, `CONFIG?` or `CONFIG=` as the answer to `CONFIG=`.

2.3 Errors

A failed command returns `ok: false` with a reason in `fout` (Dutch for "error"):

```
{"cmd":"CONFIG=","ok":false,"fout":"onbekend veld in bmp58x: presure_osr"}
{"cmd":"?","ok":false,"fout":"onbekend commando"}
```

The reason is a human-readable text for the log. Decide in your software on `ok` and the HTTP status code, not on the wording of `fout`.

2.4 Timing

Most commands answer within milliseconds. `AVG` first takes all its readings and then answers: with the default 16 readings at 50 ms that is about one second, with 200 readings about ten seconds. A timeout of **30 s** covers every command.

---

3. Commands

Command	Answer	USB	Wi-Fi
`ID`	identity: firmware and protocol version, device type, hardware identity, label	✓	✓
`STATUS`	state of the measurement: sensors present, error counters, last error, uptime	✓	✓
`SYS`	state of the electronics: memory, flash, software versions	✓	✓
`READ`	one single reading of all quantities	✓	✓
`AVG [n]`	mean, standard deviation and n over `n` readings	✓	✓
`CONFIG?`	read the measurement settings	✓	✓
`CONFIG=<json>`	change the measurement settings	✓	—
`ACCU?`	battery state	✓	✓
`TIME?`	state of the real-time clock	✓	✓
`TIME=<epoch>`	set the clock, in UTC seconds	✓	—
`LOG?`	state of the environmental log	✓	✓
`LOGLIJST`	list of log files on the microSD card	✓	✓
`LOGLEES <name> [offset]`	part of a log file, from a byte offset	✓	whole file at once
`LOGWIS <name>`	delete one log file	✓	—
`WIFI?`	Wi-Fi state	✓	—
`WIFI=<json>`	Wi-Fi settings	✓	—

Command	Answer	USB	Wi-Fi
`SCAN`	which internal components respond (diagnostics)	✓	✓
`LEDTEST`	runs the status light through red, green and blue	✓	✓ (POST)

Command and field names are part of the protocol and stay as they are, including the ones in Dutch. This document explains each one.

---

4. Measurements

4.1 READ and AVG

`READ` takes one reading. `AVG n` takes `n` readings (1 to 200) and reports the **mean** (`m`) and the **sample standard deviation** (`sd`, with n − 1) of each quantity. Without `n`, `AVG` uses the configured number of readings (16 by default).

```
> AVG 8
< {"cmd":"AVG","ok":true,"seq":2,"uptime_ms":1950,"n":8,"duur_ms":400,"sht45":{"t_c":
{"m":24.8951,"sd":0.01214},"rh":{"m":57.5007,"sd":0.01592}},"bmp58x":{"p_pa":
{"m":101170.184,"sd":3.1777},"t_c_intern":{"m":24.8751,"sd":0.01214}}}|2d1fbc18
> READ
< {"cmd":"READ","ok":true,"seq":1,"uptime_ms":1550,"sht45":{"t_c":24.91,"rh":57.5048},"bmp58x":
{"p_pa":101172.723,"t_c_intern":24.89}}|2fa6a30a
```

(Example values from the simulator, not real measurements; the CRCs are correct.)

Field	Quantity	Unit
`sht45.t_c`	air temperature	°C
`sht45.rh`	relative humidity	%RH
`bmp58x.p_pa`	barometric pressure	**Pa** (101 325 Pa = 101.325 kPa)
`bmp58x.t_c_intern`	internal temperature of the pressure sensor	°C — **diagnostic only, never use it as the air temperature**
`n`	number of readings in the mean	—
`duur_ms`	how long the averaging took	ms
`seq`	number of measurements since start-up	—
`uptime_ms`	time since start-up	ms

`sht45` and `bmp58x` are the names of the two measurement blocks: the first holds temperature and humidity, the second pressure. The names refer to the sensor types fitted in this version and are part of the protocol: if a later version uses other sensors, that comes with a new protocol version (section 9).

There is no timestamp in a measurement. Timestamp it in your software when you send the request. `seq` and `uptime_ms`, together with `boot_id` from `ID` and `STATUS`, show whether the device has restarted between two measurements: after a restart `boot_id` changes and `seq` starts again.

A failed measurement returns `ok: false` with `fout: "leesfout"` (read error). `STATUS` then tells you which sensor failed (`sht45_ok`, `bmp58x_ok`, `laatste_fout`, `i2c_fouten`).

4.2 Identifying the device

```
> ID
< {"cmd":"ID","ok":true,"fw":"0.22.0","proto":21,"dev":"aura2","hw":"feather-esp32s3-
nopsram","chip_id":"1020ba63233c","boot_id":"172fd712","module_label":"2026-0001","sht45":
{"serial_number":"0x1A2B3C4D"},"bmp58x":{"chip_id":"0x50","rev_id":"0x32"}}|ae958f56
```

Field	Meaning
`fw`	firmware version
`proto`	protocol version. **Check this** before anything else; see section 9
`dev`	device type, `"aura2"`
`chip_id`	unique hardware ID of the electronics (factory-set, cannot be changed)
`sht45.serial_number`	unique serial number of the temperature and humidity sensor
`bmp58x.chip_id`, `rev_id`	type and revision of the pressure sensor (not unique)
`module_label`	the label of this unit, e.g. the number on the device label
`boot_id`	random value that changes at every start-up

Identify a unit by `chip_id` together with `sht45.serial_number`. These are hardware facts and cannot be changed. The `module_label` is a name for people; it can be changed over USB and is not a key. Record the identity with every measurement you store, so each value can be traced to the unit and its calibration certificate.

---

5. Device state

5.1 STATUS: the state of the measurement

```
> STATUS
<
{"cmd":"STATUS","ok":true,"uptime_ms":1950,"seq":2,"boot_id":"172fd712","sht45_ok":true,"bmp58x_ok":true,"i2c
_fouten":0,"laatste_fout":null,"reset_reden":"poweron","vrij_heap":351560}|134ebe99
```

Field	Meaning
`uptime_ms`	time since start-up, ms
`seq`	number of measurements since start-up. Lines of the environmental log count as well (section 7), so a gap in `seq` between two of your own requests is the log at work
`boot_id`	random value that changes at every start-up, as in `ID`
`sht45_ok`	whether the temperature and humidity sensor was found and is in use
`bmp58x_ok`	the same for the pressure sensor
`i2c_fouten`	number of communication errors with the sensors since start-up
`laatste_fout`	why the last failed measurement failed, or `null` if none has failed since start-up. It **stays** after a successful measurement, so read it together with `i2c_fouten`
`reset_reden`	reason for the last start-up: `poweron` (power applied), `extern` (external reset), `software` (software restart), `panic` (firmware error), `int_wdt`, `task_wdt` or `wdt` (watchdog), `deepsleep`, `brownout` (supply voltage too low), `sdio`, or `onbekend` (unknown)
`vrij_heap`	free working memory, bytes; diagnostic

5.2 SYS: the state of the electronics

`SYS` describes the module as a computer, not the measurement. It is the longest answer the Aura2 sends (about 1050 characters) and mostly static, so ask for it when you need it, not with every measurement. Besides `uptime_ms` it has six blocks:

Block	Fields
`chip`	`model`; `revisie` (the major revision only); `cores`; `cpu_mhz` (the processor speed now); `chip_t_c` (temperature of the processor in °C, **never** the air temperature; `null` if not available)
`geheugen`	working memory in bytes: `heap_totaal`, `heap_vrij`, `heap_vrij_laagst` (the lowest free amount since start-up), `heap_grootste_blok` (the largest block that can be reserved at once), `psram_totaal` and `psram_vrij` (0: this model has no external RAM)
`flash`	`grootte` (bytes), `snelheid_hz`, `modus`, `app_partitie` (the firmware slot running now), `app_grootte`, `app_gebruikt`, and `ota_ruimte`: the size of the **other** firmware slot, where an update goes. It is not the free space in the running slot; that is `app_grootte` minus `app_gebruikt`
`nvs`	the store that holds the settings and the label: `gebruikt`, `vrij` and `totaal` entries, and `namespaces`. If its statistics cannot be read, there is a field `fout` instead
`partities`	the layout of the flash memory: per part `naam`, `type`, `subtype`, `adres` and `grootte`
`software`	versions of the system software under the firmware: `idf` and `arduino`

5.3 ACCU?: the battery

The battery gauge estimates the charge from the cell voltage; there is no current measurement. Treat the answer as an indication of how long the Aura2 will keep running, not as a measurement.

Field	Meaning
`meter`	type of battery gauge, or `null` if none responds
`spanning_v`	cell voltage, V
`percentage`	charge according to the gauge, %. It can be slightly above 100 and is not capped. At the low end it is too pessimistic: use `schatting`
`verandering_pct_uur`	change per hour according to the gauge, %/h; positive is charging. `null` if not available
`meter_versie`	version reported by the gauge, hexadecimal; diagnostic
`toestand`	`vol` (full), `ladend` (charging), `ontladend` (discharging), `bijna leeg` (nearly empty), `in bedrijf` (in use) or `geen meter` (no gauge)
`schatting`	estimate of the remaining charge and time from the cell voltage; see below
`lampje`	which battery warning the status light shows now: `geen` (none), `waarschuwing`, `pas op` or `kritisch` (thresholds in section 8)
`lampje_drempels`	the thresholds set for those warnings: `waarschuwing_pct`, `pas_op_pct`, `kritisch_pct`
`voeding`	`usb`, `accu` (battery) or `onbekend` (unknown): where the power **appears** to come from. Derived from the battery and the USB connection, not measured
`usb_computer`	whether a computer is connected to the USB port, as reported by the USB connection itself. A charger without a data connection does not count
`cpu_mhz`	the processor speed now; it follows the power source (section 8)
`let_op`	a note in Dutch ("note"), present only when there is something to report

If no gauge responds, the answer has only `meter` (`null`), `spanning_v` and `percentage` (`null`), `toestand` (`geen meter`) and `let_op`.

The block `schatting` (estimate):

Field	Meaning
`rest_pct`	remaining charge in whole %, read from a measured discharge curve of the cell. `null` while charging
`tempo_pct_uur`	how fast `rest_pct` falls, %/h, over the last hour. `null` until there is half an hour of data
`rest_uur`	estimated hours left: `rest_pct` divided by that rate. Only when `status` is `ok`, otherwise `null`
`minuten`	number of one-minute points the rate is based on, at most 60
`status`	`ok`, or why a field is `null`: `meet nog` (on battery, less than half an hour of data), `laadt` (charging), `aan de computer` (a computer is connected and supplies the power), `geen daling` (falling by less than half a percent per hour), `onder de curve` (below the lowest voltage of the curve; `rest_pct` is 0) or `geen spanning` (the voltage could not be read)

`rest_uur` is an estimate, not a measurement: 0 % is the lowest voltage at which the module has been seen running.

A full battery and a missing battery look the same while USB is connected: without a battery the gauge reads the USB charging voltage. The answer then carries a `let_op`. The only way to tell is to disconnect USB: if the device keeps running, the battery is in place.

5.4 SCAN: diagnostics

`SCAN` lists which internal components answer on the module's internal sensor bus (addresses `0x08` to `0x77`). `adressen` is a list of objects with `adres` (for example `"0x44"`) and `naam`, a short description of the component fitted in this version, or `onbekend` (unknown). Use it for diagnostics only; do not build on the names. The memory card is on a different bus and never appears here; `LOG?` shows whether it is present (section 7).

---

6. Wi-Fi and the HTTP interface

6.1 Switching Wi-Fi on

Wi-Fi is **off** by default and can only be configured over USB. The Aura2 supports 2.4 GHz networks.

```
> WIFI={"ssid":"Lab-2G","wachtwoord":"secret","aan":true}
< {"cmd":"WIFI?","ok":true,"aan":true,"ssid":"Lab-
2G","wachtwoord_gezet":true,"ntp_server":"pool.ntp.org","hostnaam":"2026-
0001","hostnaam_uit":"module_label","verbonden":true,"ip":"192.168.1.42","rssi":-57,"mac":"10:20:BA:63:23:3C"
,"api":"http://192.168.1.42/api/v1/","api_mdns":"http://2026-
0001.local/api/v1/","server_loopt":true,"instellen_over_http":false}|7696ea0c
```

Field in `WIFI=`	Meaning	Limit
`ssid`	network name	32 characters
`wachtwoord`	password. It can be set but is **never** read back; the answer only says `wachtwoord_gezet`	64 characters
`aan`	on (`true`) or off (`false`)	switching on requires an `ssid`

Field in `WIFI=`	Meaning	Limit
`ntp_server`	time server for the clock; empty switches NTP off	63 characters

Only the fields you send are changed. The answer is the `WIFI?` answer after the change, so its `cmd` is `WIFI?` (section 2.2). It shows the address of the API in `api` and `api_mdns`. `WIFI?` is available over USB only.

6.2 Finding the device on the network

The network name follows the `module_label`, reduced to lower-case letters, digits and hyphens (at most 24 characters): label `2026-0001` gives `http://2026-0001.local/`.

Without knowing the name, you can find every Aura on the local network segment with an mDNS (Bonjour/Zeroconf) query for the service `_aura._tcp` on port 80. Its TXT record contains:

Key	Content
`chip_id`	the unique hardware ID
`label`	the `module_label`
`fw`	firmware version
`proto`	protocol version
`api`	the API path, `/api/v1/`

Multicast does not pass routers, so discovery works within one network segment.

6.3 Endpoints

Every answer is the **same JSON** as the corresponding USB command, **without** the `|` and the CRC.

Method and path	Same as
`GET /api/v1/`	list of available paths
`GET /api/v1/id`	`ID`
`GET /api/v1/status`	`STATUS`
`GET /api/v1/sys`	`SYS`
`GET /api/v1/read`	`READ`
`GET /api/v1/avg`	`AVG`; `?n=32` for a chosen number of readings
`GET /api/v1/config`	`CONFIG?`
`GET /api/v1/accu`	`ACCU?`
`GET /api/v1/time`	`TIME?`
`GET /api/v1/log`	`LOG?`
`GET /api/v1/log/lijst`	`LOGLIJST`
`GET /api/v1/log/bestand?naam=<name>`	the whole log file as `text/csv` (the only answer that is not JSON)
`GET /api/v1/scan`	`SCAN`
`POST /api/v1/ledtest`	`LEDTEST`: the status light runs through three colours, so you can see which device you are talking to. The only action on this channel; takes just over one second

HTTP status	Meaning
200	success (`ok: true`)
400	invalid request, e.g. `n` out of range
404	unknown path (with a JSON body)
405	wrong method, e.g. `GET` on `/api/v1/ledtest`
500	device error, e.g. a sensor read error (`ok: false`)

Examples:

```
curl http://2026-0001.local/api/v1/avg?n=32
curl -O http://2026-0001.local/api/v1/log/bestand?naam=2026-09-19.csv
curl -X POST http://2026-0001.local/api/v1/ledtest
```

The device handles one request at a time, on both channels together: while it averages over 200 readings for one client, other requests wait.

---

7. The environmental log

The Aura2 also measures by itself and writes one line every 60 s (adjustable from 5 s to 24 h) to its microSD card, also on battery and without a computer.

The log is indicative. It is meant for looking back at trends, drift and interruptions. For calibration records, use measurements requested by your software (`READ` / `AVG`), which your software timestamps itself.

Files. One CSV file per day, named after the date in UTC: `2026-09-19.csv`. Lines written while the clock was not set go to `onbekend.csv` ("unknown"), with an empty time field. A date is never invented.

```
tijd_utc,klok,uptime_ms,seq,boot_id,t_c,t_c_sd,rh,rh_sd,p_pa,p_pa_sd,t_c_intern,n,chip_t_c,accu_v,accu_pct
2026-09-
10T11:34:56Z,host,165908,12,a6c79c77,24.8986,0.01449,57.5023,0.01696,101168.496,3.1264,24.8786,16,30.9,4.209,
98.6
```

Column	Content
`tijd_utc`	time in UTC (ISO 8601), or **empty** if the clock was not set
`klok`	where that time came from: `host` (set over USB), `ntp` (time server) or `niet_gezet` (not set)
`uptime_ms`, `seq`, `boot_id`	as in `READ` / `AVG`; always filled, so the order is known even without a clock
`t_c`, `t_c_sd`	air temperature, mean and SD (°C)
`rh`, `rh_sd`	relative humidity, mean and SD (%RH)
`p_pa`, `p_pa_sd`	pressure, mean and SD (Pa)
`t_c_intern`	internal temperature of the pressure sensor; diagnostic only
`n`	number of readings per line
`chip_t_c`	temperature of the processor; diagnostic only
`accu_v`, `accu_pct`	battery voltage and charge at that moment

A missing value is an **empty field**, never a zero.

Reading the log. Over Wi-Fi, download a whole file with `GET /api/v1/log/bestand?naam=...`. Over USB, use `LOGLIJST` for the file list and `LOGLEES <name> <offset>` to read a file in parts:

```
> LOGLEES 2026-09-10.csv 0
< {"cmd":"LOGLEES","ok":true,"naam":"/2026-09-10.csv","bytes":348,"offset":0,"regels":
["tijd_utc,klok,...","2026-09-10T00:26:40Z,host,..."],"volgende":348,"klaar":true}|...
```

Continue with the offset given in `volgende` until `klaar` (done) is `true`. **Always use `volgende` as given**; do not calculate offsets from the line lengths yourself (lines on the card end with CRLF, the lines in the answer are trimmed).

`LOGLIJST` answers with `kaart` (whether a card is present) and `bestanden`, a list of objects with `naam` and `bytes`. `LOGWIS <name>` (USB only) answers with `gewist`, the name of the deleted file.

The state of the log. `LOG?`:

```
> LOG?
<
{"cmd":"LOG?","ok":true,"aan":true,"interval_s":60,"kaart":true,"kaart_mb":30436,"gebruikt_mb":1,"totaal_mb":
30400,"regels":2,"bestand":"/2026-09-
10.csv","laatste_schrijf_ms":3550,"laatste_fout":null,"uptime_ms":3550}|b77d50f8
```

Field	Meaning
`aan`, `interval_s`	whether the log is on, and the interval in seconds
`kaart`	whether a microSD card is present. Without a card the answer carries a `let_op` (a note in Dutch); measurements on request still work
`kaart_mb`	size of the card, MB (`null` without a card)
`gebruikt_mb`, `totaal_mb`	used and total space of the file system on the card, MB
`regels`	number of log lines written since start-up
`bestand`	the file written to now, or `null`
`laatste_schrijf_ms`	`uptime_ms` of the last line written
`laatste_fout`	why the last write failed, or `null`
`uptime_ms`	time since start-up, ms

The clock. `TIME?` reports the time and whether it is valid:

```
> TIME?
< {"cmd":"TIME?","ok":true,"rtc":"pcf8523","geldig":true,"oscillator_gestopt":false,"tijd_utc":"2026-09-
10T00:26:40Z","epoch":1789000000,"bron":"host","gezet_utc":"2026-09-
10T00:26:40Z","zetten_over_http":false}|7f73f520
```

Field	Meaning
`rtc`	type of the clock chip, or `null` if none responds
`geldig`	whether the time can be trusted (valid). `false` means: **log lines carry no date**
`oscillator_gestopt`	the clock has stopped at some point (never set, or the backup cell ran empty); then the time is not valid
`tijd_utc`, `epoch`	the time, as ISO 8601 in UTC and in seconds since 1970; `null` if not available

Field	Meaning
`bron`	where the time came from: `host` (set over USB) or `ntp` (time server)
`gezet_utc`	when the clock was last set
`zetten_over_http`	always `false`: the clock can only be set over USB
`let_op`	a note in Dutch, present when there is no clock or its time is not valid

Set the clock over USB with `TIME=<seconds since 1970, UTC>`; values before 2020 or after 2100 are refused. A successful `TIME=` answers with the `TIME?` answer after the change (`cmd` is `TIME?`, section 2.2). With Wi-Fi on, the device also sets its clock from the time server when it connects and every six hours. The device knows only UTC.

8. Settings

`CONFIG?` reads the measurement settings; `CONFIG=` changes them, **over USB only**. The settings are stored in the device and survive a restart. A change is applied **completely or not at all**, and only the fields you send are changed.

```
{
  "module_label": "2026-0001",
  "sht45":  {"precision": "high", "heater": "off", "n_software_averages": 16, "sample_interval_ms": 50},
  "bmp58x": {"pressure_osr": 16, "temperature_osr": 1, "iir_filter_coef": 0, "read_mode": "forced",
             "n_software_averages": 16, "sample_interval_ms": 50},
  "logboek": {"aan": true, "interval_s": 60},
  "accu_lampje": {"waarschuwing_pct": 20, "pas_op_pct": 10, "kritisch_pct": 5},
  "cpu": {"usb_mhz": 240, "accu_mhz": 80},
  "i2c_speed_hz": 100000
}
```

The blocks `accu_lampje` (when the status light warns of a low battery) and `cpu` (processor speed on USB and on battery power) are device settings, not measurement settings; they do not affect the measurement. A successful `CONFIG=` answers with the `CONFIG?` answer after the change (`cmd` is `CONFIG?`, section 2.2), so you see what the device actually has.

Setting	Allowed	Why
`module_label`	at most 32 characters	refused, not truncated
`n_software_averages`	1 – 200, the same for both blocks	both sensors are read in one pass
`sample_interval_ms`	0 – 10 000, the same for both blocks	idem
`pressure_osr`	1, 2, 4, 8, 16 or 32	free choice; 16 by default
`heater`	`off` **only**	heating would raise the temperature and lower the humidity
`temperature_osr`	**1 only**	limits self-heating
`iir_filter_coef`	**0 only**	no smoothing of the sensor data
`read_mode`	`forced` **only**	one fresh reading per request
`logboek.aan`	`true` / `false`	log on or off (on by default)
`logboek.interval_s`	5 – 86 400	log interval in seconds
`i2c_speed_hz`	100 000 or 400 000	internal bus speed

Setting	Allowed	Why
`accu_lampje.waarschuwing_pct`, `pas_op_pct`, `kritisch_pct`	0 – 100 each, **descending** (`kritisch_pct` ≤ `pas_op_pct` ≤ `waarschuwing_pct`); 0 switches that level off. Default 20, 10, 5	below these remaining charges the status light blinks red at 1, 2 or 3 Hz while the battery is not charging. The frequencies are fixed; a non-descending set is refused, not sorted
`cpu.usb_mhz`, `cpu.accu_mhz`	80, 160 or 240 each. Default 240 and 80	processor speed on USB power and on battery power. The measurement does not depend on it; a lower speed saves battery and warms the module less

Settings that affect comparability are locked: the device refuses other values **and says why**. An unknown field is refused as well, so a typing error in a field name does not pass silently.

Note: a `CONFIG?` answer contains the protocol fields `cmd` and `ok`. Remove them before sending the settings back with `CONFIG=`.

9. Versions and compatibility

- `proto` in the `ID` answer is the version of the USB protocol and its JSON fields. **Check it when you connect**, and stop if it is not a version your software knows. Within one version, the commands and fields do not change.
- The `v1` in `/api/v1/` is the version of the HTTP paths. It only changes if the paths change.
- `fw` is the firmware version. A firmware update can change behaviour without changing the protocol; the protocol version only goes up when commands or fields change.

This document always describes the current protocol version.

10. Example: reading the Aura2 from Python

Over USB, with only `pyserial` and the standard library:

```
import json
import zlib

import serial  # pip install pyserial


def ask(port, command):
    port.write(f"{command}\n".encode())
    line = port.readline().decode().strip()
    body, _, crc = line.rpartition("|")
    if zlib.crc32(body.encode()) != int(crc, 16):
        raise IOError(f"CRC mismatch on the answer to {command!r}")
    answer = json.loads(body)
    if not answer["ok"]:
        raise IOError(answer.get("fout", "error"))
    return answer


with serial.Serial("COM5", 115200, timeout=30) as port:  # pyserial sets DTR by default
    identity = ask(port, "ID")
    assert identity["dev"] == "aura2" and identity["proto"] == 21
    m = ask(port, "AVG 16")
    print(m["sht45"]["t_c"]["m"], "°C", m["sht45"]["rh"]["m"], "%RH", m["bmp58x"]["p_pa"]["m"], "Pa")
```

Over Wi-Fi, with the standard library only:

```
import json
import urllib.request

with urllib.request.urlopen("http://2026-0001.local/api/v1/avg?n=16", timeout=30) as r:
    m = json.load(r)
print(m["sht45"]["t_c"]["m"], m["sht45"]["rh"]["m"], m["bmp58x"]["p_pa"]["m"])
```