

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.eu · nexchrom.nl · 5 October 2026

CONTENTS

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

Applies to protocol version **20**, firmware **0.21.0** 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 <code>/api/v1/</code>
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	<code>id</code> works the same as <code>ID</code>
Leading and trailing whitespace is ignored	so a <code>\r</code> 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.

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
<code>ID</code>	identity: firmware and protocol version, device type, hardware identity, label	✓	✓
<code>STATUS</code>	state of the measurement: sensors present, error counters, last error, uptime	✓	✓
<code>SYS</code>	state of the electronics: memory, flash, software versions	✓	✓
<code>READ</code>	one single reading of all quantities	✓	✓
<code>AVG [n]</code>	mean, standard deviation and n over <code>n</code> readings	✓	✓
<code>CONFIG?</code>	read the measurement settings	✓	✓
<code>CONFIG=<json></code>	change the measurement settings	✓	—
<code>ACCU?</code>	battery state	✓	✓
<code>TIME?</code>	state of the real-time clock	✓	✓
<code>TIME=<epoch></code>	set the clock, in UTC seconds	✓	—
<code>LOG?</code>	state of the environmental log	✓	✓
<code>LOGLIST</code>	list of log files on the microSD card	✓	✓
<code>LOGLEES <name> [offset]</code>	part of a log file, from a byte offset	✓	whole file at once
<code>LOGWIS <name></code>	delete one log file	✓	—
<code>WIFI?</code>	Wi-Fi state	✓	—
<code>WIFI=<json></code>	Wi-Fi settings	✓	—
<code>SCAN</code>	which internal components respond (diagnostics)	✓	✓
<code>LEDTEST</code>	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
<code>sht45.t_c</code>	air temperature	°C
<code>sht45.rh</code>	relative humidity	%RH
<code>bmp58x.p_pa</code>	barometric pressure	Pa (101 325 Pa = 101.325 kPa)
<code>bmp58x.t_c_intern</code>	internal temperature of the pressure sensor	°C — diagnostic only, never use it as the air temperature
<code>n</code>	number of readings in the mean	—
<code>duur_ms</code>	how long the averaging took	ms
<code>seq</code>	number of measurements since start-up	—
<code>uptime_ms</code>	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.

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":"0.21.0","proto":20,"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"}}|ff3caa7a
```

Field	Meaning
<code>fw</code>	firmware version
<code>proto</code>	protocol version. Check this before anything else; see section 8
<code>dev</code>	device type, <code>"aura2"</code>
<code>chip_id</code>	unique hardware ID of the electronics (factory-set, cannot be changed)
<code>sht45.serial_number</code>	unique serial number of the temperature and humidity sensor
<code>bmp58x.chip_id, rev_id</code>	type and revision of the pressure sensor (not unique)
<code>module_label</code>	the label of this unit, e.g. the number on the device label
<code>boot_id</code>	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. Wi-Fi and the HTTP interface

5.1 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,"installen_over_http":false}|7696ea0c
```

Field in <code>WIFI=</code>	Meaning	Limit
<code>ssid</code>	network name	32 characters
<code>wachtwoord</code>	password. It can be set but is never read back; the answer only says <code>wachtwoord_gezet</code>	64 characters
<code>aan</code>	on (<code>true</code>) or off (<code>false</code>)	switching on requires an <code>ssid</code>
<code>ntp_server</code>	time server for the clock; empty switches NTP off	63 characters

Only the fields you send are changed. The answer shows the address of the API in `api` and `api_mdns`.

5.2 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.

5.3 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/lijt	LOGLIJT
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.

6. 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
<code>tijd_utc</code>	time in UTC (ISO 8601), or empty if the clock was not set
<code>klok</code>	where that time came from: <code>host</code> (set over USB), <code>ntp</code> (time server) or <code>niet_gezet</code> (not set)
<code>uptime_ms</code> , <code>seq</code> , <code>boot_id</code>	as in <code>READ / AVG</code> ; always filled, so the order is known even without a clock
<code>t_c</code> , <code>t_c_sd</code>	air temperature, mean and SD (°C)
<code>rh</code> , <code>rh_sd</code>	relative humidity, mean and SD (%RH)
<code>p_pa</code> , <code>p_pa_sd</code>	pressure, mean and SD (Pa)
<code>t_c_intern</code>	internal temperature of the pressure sensor; diagnostic only
<code>n</code>	number of readings per line
<code>chip_t_c</code>	temperature of the processor; diagnostic only
<code>accu_v</code> , <code>accu_pct</code>	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).

The clock. `TIME?` reports the time and whether it is valid (`ge1dig`). Set it over USB with `TIME=<seconds since 1970, UTC>`; values before 2020 or after 2100 are refused. 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.

7. 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},
  "i2c_speed_hz": 100000
}
```

Two further 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.

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

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=`.

8. 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.

9. 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"] == 20
    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"])
```