API - Data Retrieval

- How to retrieve data from the Whysor API

How to retrieve data from the Whysor API

Whysor API - Device & Sensor Data

Overview

This document describes how to retrieve devices, sensors, and sensor measurements using the Whysor API, reachable at **https://api.whysor.com**.

This API guide explains how users can:

- Log in and obtain an access token
- Retrieve devices they have access to (single device, list, or filtered list)
- Retrieve sensors belonging to devices (single sensor, list, or filtered list)
- Read the latest/historical measurement data for a single sensor
- Read measurement data for multiple sensors in one request

All API requests below (except login) require authentication.

Authentication

Log in

Request

```
POST /users/login
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "your-password"
}
```

If you're logging in through a white-labeled deployment, pass that deployment's own domain instead, e.g.:

```
{
  "email": "user@example.com",
  "password": "your-password",
  "domain": "my.example.org"
}
```

`domain` selects which login realm to authenticate against. Whysor is white-labeled — the same email can exist under different organizations, each tied to its own domain, and the `domain` parameter tells the server which one to log you into. whysor.com is the default realm, used automatically if `domain` is omitted or doesn't match any known theme's domain.

Response `200 OK`

```
{
  "id": "<access-token>",
  "ttl": 1209600,
  "created": "2026-09-15T09:12:03.000Z",
  "userId": "usr-0001-mock"
}
```

The `id` field is your access token. It is valid for `ttl` seconds (14 days by default). The examples below use `<token>` as a placeholder for this value.

Authenticate subsequent requests

Pass the token in the `Authorization` header on every request (no `Bearer` prefix):

```
Authorization: <token>
Content-Type: application/json
```

If the header is missing or invalid, every endpoint returns:

```
{ "error": { "statusCode": 401, "name": "Error", "message": "Authorization Required", "code": "AUTHORIZATION_REQUIRED" } }
```

Log out

Request

```
POST /users/logout
Authorization: <token>
```

Response `204 No Content` (empty body — the token is invalidated).

Devices

Get all devices you have access to

Request

```
GET /devices
Authorization: <token>
```

Response `200 OK`


```
[
  {
    "id": "dev-0001-mock",
    "active": true,
    "externalDomain": "visual-crossing",
    "externalId": "ext-0001-mock",
    "additionalId": "000001",
    "name": "Mock Weather Station 1",
    "description": "",
    "additionalInfo": { "location": { "latitude": 51.0, "longitude": 6.0 } },
    "deleted": false,
    "created": "2024-06-10T12:47:18.669Z",
```

```
    "modified": "2025-05-23T09:35:17.438Z",
    "organizationId": "org-0001-mock"
  }
]
```

(Additional device objects in the array are truncated here for brevity — each has the same shape.)

Get devices with a filter

The `filter` query parameter takes a JSON-encoded LoopBack-style filter object supporting `where`, `fields`, `include`, `order`, `limit`, and `offset`.

Request — devices with a name matching "Mock" (case-insensitive)

```
GET /devices?filter={"where":{"name":{"like":"Mock","options":"i"}}}
Authorization: <token>
```

Response `200 OK`

```
[
  {
    "id": "dev-0001-mock",
    "active": true,
    "externalDomain": "visual-crossing",
    "externalId": "ext-0001-mock",
    "additionalId": "000001",
    "name": "Mock Weather Station 1",
    "description": "",
    "additionalInfo": { "location": { "latitude": 51.0, "longitude": 6.0 } },
    "deleted": false,
    "created": "2024-06-10T12:47:18.669Z",
    "modified": "2025-05-23T09:35:17.438Z",
    "organizationId": "org-0001-mock"
  }
]
```

Get a single device by ID

Request

```
GET /devices/dev-0001-mock
Authorization: <token>
```

Response `200 OK`


```
{
  "id": "dev-0001-mock",
  "active": true,
  "externalDomain": "visual-crossing",
  "externalId": "ext-0001-mock",
  "additionalId": "000001",
  "name": "Mock Weather Station 1",
  "description": "",
  "additionalInfo": { "location": { "latitude": 51.0, "longitude": 6.0 } },
  "deleted": false,
  "created": "2024-06-10T12:47:18.669Z",
  "modified": "2025-05-23T09:35:17.438Z",
  "organizationId": "org-0001-mock"
}
```


Find the first device matching a filter

Request

```
GET /devices/findOne?filter={"where":{"externalId":"ext-0001-mock"}}
Authorization: <token>
```

Response `200 OK` — a single object, same shape as above (not an array).

If no device matches, the API returns `404`:


```
{ "error": { "statusCode": 404, "name": "Error", "message": "Unknown \"Device\" id
\"undefined\".", "code": "MODEL_NOT_FOUND" } }
```


Count devices matching a filter

Request

```
GET /devices/count?where={"active":true}
Authorization: <token>
```

Response `200 OK`

```
{ "count": 3 }
```

Sensors

Get all sensors you have access to

Request

```
GET /sensors
Authorization: <token>
```

Response `200 OK` — an array of sensor objects (see shape below).

Get sensors for a specific device

Use `where.deviceId` in the filter.

Request

```
GET /sensors?filter={"where":{"deviceId":"dev-0001-mock"}}
Authorization: <token>
```

Response `200 OK`

```
[
  {
    "id": "sen-0001-mock",
    "name": "",
    "description": "",
    "tag": "humidity",
    "hasForecast": true,
    "virtual": false,
```

```
    "additionalInfo": {},
    "calibrations": [],
    "deviceId": "dev-0001-mock",
    "created": "2024-06-10T12:47:18.814Z",
    "modified": "2024-06-10T12:47:18.814Z"
  },
  {
    "id": "sen-0002-mock",
    "name": "",
    "description": "",
    "tag": "temperature",
    "hasForecast": true,
    "virtual": false,
    "additionalInfo": {},
    "calibrations": [],
    "deviceId": "dev-0001-mock",
    "created": "2024-06-10T12:47:18.813Z",
    "modified": "2024-06-10T12:47:18.814Z"
  }
]
```


Get sensors with other filters

Any Sensor field can be used in `where` (e.g. `tag`, `name`, `sensorTemplateId`). This is **not** scoped to one device — it matches across all devices you have access to.

Request

```
GET /sensors?filter={"where":{"tag":"temperature"},"limit":3}
Authorization: <token>
```

Response `200 OK`


```
[
  {
    "id": "sen-0004-mock",
    "tag": "temperature",
    "deviceId": "dev-0002-mock",
    "hasForecast": true,
    "virtual": false,
```

```
    "additionalInfo": {},
    "calibrations": [],
    "name": "",
    "description": "",
    "created": "2024-06-10T13:12:59.478Z",
    "modified": "2024-06-10T13:12:59.478Z"
  },
  {
    "id": "sen-0002-mock",
    "tag": "temperature",
    "deviceId": "dev-0001-mock",
    "hasForecast": true,
    "virtual": false,
    "additionalInfo": {},
    "calibrations": [],
    "name": "",
    "description": "",
    "created": "2024-06-10T12:47:18.813Z",
    "modified": "2024-06-10T12:47:18.814Z"
  },
  {
    "id": "sen-0005-mock",
    "tag": "temperature",
    "deviceId": "dev-0003-mock",
    "hasForecast": true,
    "virtual": false,
    "additionalInfo": {},
    "calibrations": [],
    "name": "",
    "description": "",
    "created": "2024-06-07T12:54:47.520Z",
    "modified": "2024-06-07T12:54:47.520Z"
  }
]
```


Combine device and other filters:

Request

```
GET /sensors?filter={"where":{"deviceId":"dev-0001-mock","tag":"temperature"}}
Authorization: <token>
```

Response `200 OK`


```
[
  {
    "id": "sen-0002-mock",
    "tag": "temperature",
    "deviceId": "dev-0001-mock",
    "hasForecast": true,
    "virtual": false,
    "additionalInfo": {},
    "calibrations": [],
    "name": "",
    "description": "",
    "created": "2024-06-10T12:47:18.813Z",
    "modified": "2024-06-10T12:47:18.814Z"
  }
]
```


Get a single sensor by ID

Request

```
GET /sensors/sen-0002-mock
Authorization: <token>
```

Response `200 OK`


```
{
  "id": "sen-0002-mock",
  "name": "",
  "description": "",
  "tag": "temperature",
  "hasForecast": true,
  "virtual": false,
  "additionalInfo": {},
  "calibrations": [],
  "deviceId": "dev-0001-mock",
  "created": "2024-06-10T12:47:18.813Z",
  "modified": "2024-06-10T12:47:18.814Z"
```

```
}
```

Find the first sensor matching a filter

Request

```
GET /sensors/findOne?filter={"where":{"deviceId":"dev-0001-mock"}}
Authorization: <token>
```

Response `200 OK` — a single object, same shape as above (not an array).

Count sensors matching a filter

Request

```
GET /sensors/count?where={"deviceId":"dev-0001-mock"}
Authorization: <token>
```

Response `200 OK`

```
{ "count": 21 }
```

Sensor Data

There are two ways to read measurement data: one sensor at a time, or many sensors in a single request.

Read data for a single sensor

Request

```
GET /sensors/sen-0002-mock/read?filter={"limit":3,"order":"datetimeMeasure DESC"}
Authorization: <token>
```

Response `200 OK`

```
[
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T07:00:00.000Z", "value": 17.7,
"groupId": null, "metadata": null },
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T05:00:00.000Z", "value": 15.2,
"groupId": null, "metadata": null },
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T04:00:00.000Z", "value": 15.7,
"groupId": null, "metadata": null }
]
```

The `filter` supports the usual `where`, `order`, `limit`, `offset` options, applied against the measurement records (e.g. filter by `datetimeMeasure` using `gt`, `lt`, or `between`).

Request — a time range

```
GET /sensors/sen-0002-mock/read?filter={"where":{"datetimeMeasure":{"between":["2026-09-
14T00:00:00.000Z","2026-09-15T00:00:00.000Z"]}},"order":"datetimeMeasure ASC"}
Authorization: <token>
```

Response `200 OK` — returned 19 measurements for that 24-hour window (array truncated here):

```
[
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-14T01:00:00.000Z", "value": 16.5,
"groupId": null, "metadata": null },
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-14T03:00:00.000Z", "value": 16.3,
"groupId": null, "metadata": null }
]
```

Downsampling large ranges (decimation)

For large time ranges, you can ask the API to downsample the results using the `lttb` (Largest-Triangle-Three-Buckets) method instead of returning every raw point.

Request

```
GET /sensors/sen-0002-mock/read?filter={"where":{"datetimeMeasure":{"between":["2026-08-
01T00:00:00.000Z","2026-09-15T00:00:00.000Z"]}},"decimate":{"method":"lttb","samples":10}}
Authorization: <token>
```

Response `200 OK` — 13 downsampled points covering the ~6-week range (`samples` is a target, not an exact count):

```
[
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-08-01T00:00:00.000Z", "value": 18.1,
"groupId": null, "metadata": null },
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-08-03T15:00:00.000Z", "value": 34.8,
"groupId": null, "metadata": null },
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-08-07T04:00:00.000Z", "value": 8.8,
"groupId": null, "metadata": null },
  { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-14T23:00:00.000Z", "value": 16.7,
"groupId": null, "metadata": null }
]
```


`samples` is the target number of points to return (defaults to 50 if omitted).

Read data for multiple sensors at once

Request — no filter (returns only the latest measurement per sensor)

```
POST /sensors/read
Authorization: <token>
Content-Type: application/json

{
  "sensorIds": [
    "sen-0002-mock",
    "sen-0001-mock"
  ]
}
```

Response `200 OK`


```
{
  "sen-0002-mock": [
    { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T07:00:00.000Z", "value":
17.7, "groupId": null, "metadata": null }
  ],
  "sen-0001-mock": [
    { "sensorId": "sen-0001-mock", "datetimeMeasure": "2026-09-15T07:00:00.000Z", "value": 95,
"groupId": null, "metadata": null }
  ]
}
```

Response is an object keyed by `sensorId` (not an array like the single-sensor endpoint), each value an array of measurements for that sensor.

Filtering the multi-sensor read

You can include a `filter` alongside `sensorIds` for a time range or decimation. **This must go in the JSON body, not the query string** — passing `?filter=...` on this endpoint conflicts with the body and the server responds `422`:

```
{ "error": { "statusCode": 422, "name": "InvalidRequestError", "message": "At least one sensor ID is required.", "code": "INVALID_REQUEST" } }
```

The correct way — put `filter` as a sibling key of `sensorIds`:

Request

```
POST /sensors/read
Authorization: <token>
Content-Type: application/json

{
  "sensorIds": ["sen-0002-mock"],
  "filter": {
    "where": { "datetimeMeasure": { "gt": "2026-09-15T00:00:00.000Z" } },
    "order": "datetimeMeasure ASC"
  }
}
```

Response `200 OK`

```
{
  "sen-0002-mock": [
    { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T02:00:00.000Z", "value": 16.1, "groupId": null, "metadata": null },
    { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T03:00:00.000Z", "value": 16.0, "groupId": null, "metadata": null },
    { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T04:00:00.000Z", "value": 15.7, "groupId": null, "metadata": null },
    { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T05:00:00.000Z", "value": 15.2, "groupId": null, "metadata": null },
    { "sensorId": "sen-0002-mock", "datetimeMeasure": "2026-09-15T07:00:00.000Z", "value": 17.7, "groupId": null, "metadata": null }
```

```
  ]
}
```

Errors

At least one sensor ID is required — an empty array is rejected:

Request

```
POST /sensors/read
Authorization: <token>
Content-Type: application/json

{ "sensorIds": [] }
```

Response `422 Unprocessable Entity`

```
{ "error": { "statusCode": 422, "name": "InvalidRequestError", "message": "At least one sensor ID is required.", "code": "INVALID_REQUEST" } }
```

Only sensors your token has access to, whose device is active, will be included in the response — sensor IDs you don't have permission for are silently omitted rather than causing an error.

Putting it together: fetch a device's sensors and their latest data

```
# 1. Log in
POST /users/login
{ "email": "user@example.com", "password": "your-password" }
→ token = response.id

# 2. Find the device
GET /devices/findOne?filter={"where":{"externalId":"ext-0001-mock"}}
Authorization: <token>
→ deviceId = response.id  // "dev-0001-mock"

# 3. Get its sensors
```

```
GET /sensors?filter={"where":{"deviceId":"dev-0001-mock"},"limit":2}
Authorization: <token>
→ sensorIds = response.map(s => s.id)  // ["sen-0001-mock", "sen-0003-mock"]

# 4. Read latest data for all sensors in one call
POST /sensors/read
Authorization: <token>
{ "sensorIds": ["sen-0001-mock", "sen-0003-mock"] }
```

Example response from step 4 (values/shape verified live, IDs mocked):


```
{
  "sen-0001-mock": [
    { "sensorId": "sen-0001-mock", "datetimeMeasure": "2026-09-15T07:00:00.000Z", "value": 95,
"groupId": null, "metadata": null }
  ],
  "sen-0003-mock": [
    { "sensorId": "sen-0003-mock", "datetimeMeasure": "2026-09-15T07:00:00.000Z", "value": 0,
"groupId": null, "metadata": null }
  ]
}
```