EPORDO Connect API Guide (English)

Item	Content
Document version	**1.1.5**
Release date	2026-11-11
Audience	Integrator developers, resellers / SIs, project delivery staff
Product	**EPORDO Connect** (third-party device cloud access platform)
Delivery	This Markdown source + a portal-downloadable **PDF** (Help → Download API Guide)

> This document describes the **tenant-side REST API** (path prefix `/connect/v1`). Device-to-cloud communication is handled by Connect; integrators **do not** need to parse or implement device raw binary / text protocols.

> Examples below use production-style Base URLs by default; for lab / UAT, replace with the address provided for your environment.

> **Convention:** Paths below are relative to the Base URL. JSON field names in this document are authoritative (for example, the event business body field is `payload_json`, which you must `JSON.parse` yourself).

Revision history

Version	Date	Notes
1.1.5	2026-11-11	§5.3.1 adds `door_mode` and `door_opened` in all five languages; `photo_status` is `absent`, `stored`, `omitted_too_large`, or `omitted_invalid`
1.1.4	2026-11-10	§5.3 current types are `door.remote.recorded` and `device.presence`; notes the 10-minute online window, ~5-minute offline scan, and that legacy type names match only the corresponding edge
1.1.3	2026-11-08	Polish: §5.3 column header `event_type` and add `door.open.recorded`; Appendix D command cheat-sheet uses `GET /commands/{id}` + `POST .../cancel`
1.1.2	2026-11-08	Corrections: event field is `event_type` (not `type`); photo endpoint returns JSON/ `data_base64` (not a raw image stream); no `GET /webhooks/{id}`; no `DELETE /device-groups/{id}` (use `status=DISABLED`); retention days aligned to 3–14 days
1.1.1	2026-11-08	Cross-checked against production: added API Key / group paths and examples; Webhook signature verification and outbox; event `payload_json` samples and retention notes; fan-out set-user example; table of contents
1.1.0	2026-11-08	Added system overview and communication relationships; cURL examples and scenario notes in each chapter; device unbind; revision history; PDF delivery
1.0.0	2026-11-07	First public guide: claim, events, Webhook/MQTT, canonical commands, fan-out, error-code summary

Later typo-only fixes will add a patch version in this table; behavior changes will bump the minor version and be noted in the relevant sections.

Table of contents

1. System overview
2. Environment and authentication (includes API Key management)
3. Unified error format
4. Devices
5. Events
6. Webhook / MQTT
7. Canonical commands
8. Fan-out by serial-number list
9. Device groups and Scoped Keys
10. Rate limits and daily quotas
11. Recommended integration flow

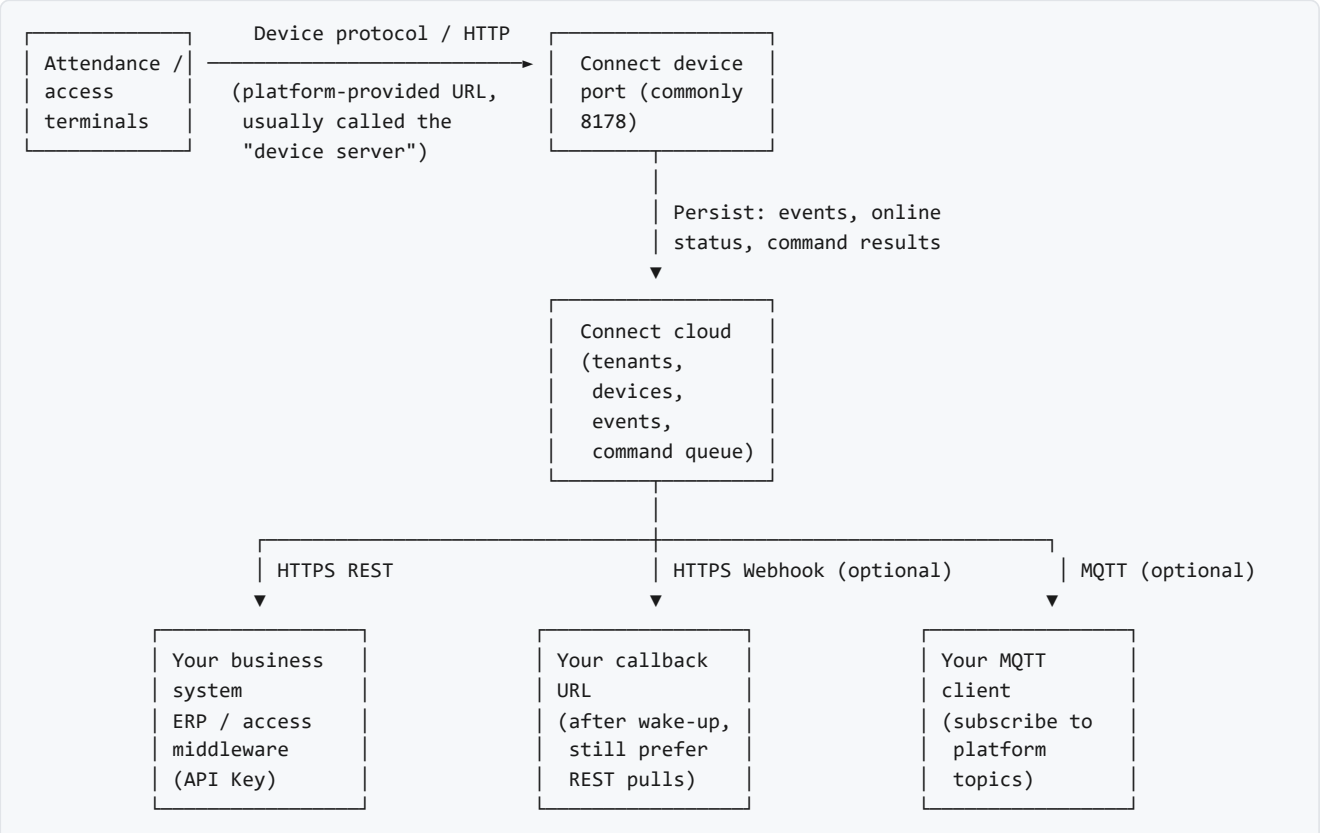
- Appendix A Error codes · Appendix B Credential kinds · Appendix C Support · Appendix D API quick reference

---

1. System overview (read this first)

1.1 Where Connect sits in the architecture

Think of it as a cloud adaptation layer between devices and your business systems:



Do not mix the two lines:

Line	Who connects to whom	What you do
Device line	Terminal → Connect device server (example: `http://api.bnetlink.com:8178/`)	Enter the platform-provided address in the device menu; **do not** assemble device frames in your code
Business line	Your servers → Connect REST (example: `https://api.bnetlink.com/connect/v1`)	Call the APIs in this guide with a **Bearer API Key**

The portal (browser) is for staff to claim devices, view events, and try commands; **for production integration use your own backend + API Key**. Do not embed the Key in a public web page.

1.2 Device communication protocols (integrator view)

Terminals talk to Connect's **device server** using vendor device protocols (for example dynamic-machine BS, ordinary-machine FK). Connect is responsible for:

1. Recognizing the serial number and owning tenant;
2. Normalizing punches / enrollment uploads into a unified **event JSON**;
3. Translating the **canonical commands** you issue via REST into device-executable frames, queuing them, and writing results back.

Therefore this guide uses canonical names such as `set-user` and `delete-user`, **not** device-internal command-code strings. You only need to:

- Claim serial number `sn`;
- Pull events / receive Webhook or MQTT;
- Issue canonical commands and poll `command_id` status.

1.3 Typical lifecycle (example)

Example: a store installs a new face terminal and connects it to your attendance middleware.

1. At factory, the vendor already wrote that serial into "sellable stock" and bound it to your tenant.
2. You (or the portal) call `POST /devices` with `sn=ABC123` → **claim succeeds**.
3. In device settings, enter the device-server URL and go online → heartbeats / uploads begin.
4. An employee faces the device → Connect emits a `punch.recorded` event → you can pull it with `GET /events`, or be woken by Webhook/MQTT first and then pull.
5. From ERP you push employee `1001`: `POST /devices/ABC123/commands/set-user`.
6. Poll `GET /commands/{command_id}` until `SUCCEEDED` or `FAILED`.
7. If the store removes the terminal: `DELETE /devices/ABC123` **unbinds**; the serial returns to unclaimed and can be claimed again later; historical punch events are retained.

1.4 What Connect does / does not do

Does:

- Claim / disable / enable / unbind devices;
- Receive punch, enrollment-change, and similar events;
- Issue canonical commands (time sync, unlock, users, access schedules, etc.);
- Optional Webhook and optional MQTT push.

Does not:

- Scheduling, payroll, or attendance-rule engines (that belongs to your business system or other products);
- Persist "which event you have read" cursors for you (store them in your own DB);
- Automatically convert face / fingerprint template formats across firmware.

1.5 The three areas integrators care about day to day

Capability	Description
Devices	Claim `sn`, list, disable / enable, unbind
Events	Pull; optional Webhook and/or MQTT for wake-up or side channel
Commands	Time sync, status query, unlock, users, schedules; optional fan-out by serial list

---

2. Environment and authentication

2.1 Base URL

Environment	REST prefix (example)
Production	Provided by contract / delivery, shaped like `https://api.bnetlink.com/connect/v1`
Local integration (if enabled)	`http://127.0.0.1:8080/connect/v1`

Paths below are relative to this prefix. For example `GET /me` = `GET {base}/me`.

Device server (for terminals) example: `http://api.bnetlink.com:8178/` (as delivered).

2.2 Authentication

Except for `GET /health`, every request must include:

```
Authorization: Bearer {API_KEY}
Content-Type: application/json
Accept: application/json
```

Notes:

- One API Key maps to one **tenant** and can only access that tenant's data.
- Keys are created on the portal "Credentials" page, or issued when the account is opened; the **plaintext appears only once**—store it safely.
- The **Webhook signing secret** and **MQTT subscription password** are **not** the same string as the API Key.
- Portal login accounts (username + password) are for browser use only and **cannot** replace an API Key for REST.

2.3 Health check (example)

```
curl -s "https://api.bnetlink.com/connect/v1/health"
```

Success:

```
{ "status": "ok" }
```

2.4 Current tenant (example)

```
curl -s "https://api.bnetlink.com/connect/v1/me" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

A successful response includes `tenant_id`, plan, Key scope, and related fields (exact fields as returned).

Example: If `scope_type` is group scope (`GROUPS`), some account-level APIs (claim, unbind, manage Webhooks / credentials / groups) return `403` `forbidden_scope`.

2.5 API Key management (REST)

The portal "Credentials" page is backed by the following APIs. On create, the plaintext is **returned only once**.

```
# List (no plaintext)
curl -s "https://api.bnetlink.com/connect/v1/credentials" \
  -H "Authorization: Bearer YOUR_API_KEY"

# Create an account-wide Key
curl -s -X POST "https://api.bnetlink.com/connect/v1/credentials" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"name\":\"erp-prod\",\"scope_type\":\"ACCOUNT\"}"

# Create a group-scoped Key (group_ids must be ACTIVE groups of this tenant)
curl -s -X POST "https://api.bnetlink.com/connect/v1/credentials" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"name\":\"store-a\",\"scope_type\":\"GROUPS\",\"group_ids\":[1,2]}"

# Disable / re-enable (status: REVOKED | ACTIVE)
curl -s -X PUT "https://api.bnetlink.com/connect/v1/credentials/2" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"status\":\"REVOKED\"}"
```

Successful create example (save `api_key` immediately):

```
{
  "api_key": "epc_live_…",
  "key_prefix": "epc_live_…",
  "scope_type": "ACCOUNT",
  "warning": "Store api_key now; it will not be shown again."
}
```

Note: you cannot disable the tenant's **last** `ACTIVE` Key (`409` `last_active_api_key`). Group Keys **cannot** log into the portal.

---

3. Unified error format

```
{
  "error": {
    "code": "device_not_found",
    "message_key": "connect.error.device_not_found",
    "message": "device_not_found"
  }
}
```

HTTP	Meaning (summary)
400	Invalid parameters
401	Missing Key / Key invalid or disabled

403	Insufficient permission (for example a group Key called an account-level API)
404	Resource missing or not owned by this tenant
409	Conflict (duplicate claim, device disabled, idempotency conflict, etc.)
429	Per-minute rate limit or daily command quota exhausted

Example: Issuing unlock to a disabled device → HTTP `409`, `code` = `device_disabled`.

Common `code` values are listed in Appendix A.

---

4. Devices

4.1 Claim a device

```
POST /devices
```

```
{ "sn": "TEST-SN-001", "alias": "Front gate" }
```

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"sn\":\"TEST-SN-001\",\"alias\":\"Front gate\"}"
```

Success example:

```
{
  "sn": "TEST-SN-001",
  "connect_device_id": 12,
  "register_status": "ACTIVE",
  "protocol_track": "BS",
  "model_code": "P5",
  "firmware_version": "1.02"
}
```

Notes:

- You may only claim serials that are in **vendor stock, already bound to this tenant, and still unclaimed**.
- Protocol / model / firmware come from vendor pre-provisioning; the request body **cannot** forge them. The model is the short sales name `P5` or `P7`. Header `dev_model` does not change it.
- Unknown serial → `404` `device_not_in_stock`.
- Assigned to another tenant → `403` `device_not_for_tenant`.
- Already claimed by this tenant → `409` `device_already_registered`.
- External field name is always `sn`.

Example: The box label shows `B515…`, but the vendor has not written that serial into your stock yet → claim returns `device_not_in_stock`; contact the vendor to pre-provision, then retry.

4.2 Device list / detail

```
curl -s "https://api.bnetlink.com/connect/v1/devices?limit=50" \
  -H "Authorization: Bearer YOUR_API_KEY"

curl -s "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

List supports cursor pagination: `next_cursor` is opaque; pass it unchanged as `cursor=` for the next page.

4.3 Disable / enable

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/disable" \
  -H "Authorization: Bearer YOUR_API_KEY"

curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/enable" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

- **Disable**: still belongs to this account, but commands can no longer be issued (`409` `device_disabled`). Device-uploaded events can usually still be persisted (per registry).
- **Enable**: restores the ability to issue commands.
- Disable ≠ unbind.

4.4 Unbind a device

```
curl -s -X DELETE "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Success example:

```
{ "sn": "TEST-SN-001", "claim_status": "UNCLAIMED" }
```

Effects:

- Device is removed from this account; vendor stock returns to "unclaimed" and **can be claimed again**;
- Platform-side enrollment for that unit is cleared; queued commands are cancelled;
- **Historical punch events are retained**;
- Group-scoped child Keys **cannot** call this (`403`).

4.5 Users and credentials on a device (current platform-side enrollment)

Method	Path	Description	
GET	`/devices/{sn}/users`	User list; optional `scope=device\`	compatible `,` q`
GET	`/devices/{sn}/users/{userId}`	User detail and credential metadata	
GET	`/devices/{sn}/users/{userId}/credentials/{kind}/{slot}`	Credential body (`data_base64`)	

```
curl -s "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/users?limit=50" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

`kind` allows only: `photo` , `face` , `fingerprint` , `palm` , `card` , `password` (see Appendix B).

5. Events

Connect **does not** store your read cursor. Record the last processed `event_id` / time in your own database.

5.1 Pull events

```
curl -s "https://api.bnetlink.com/connect/v1/events?limit=50" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Pull the last 1 day by ingest time (example; replace with real ISO-8601):

```
curl -s "https://api.bnetlink.com/connect/v1/events?received_from=2026-11-07T00:00:00Z&received_to=2026-11-08T0
  -H "Authorization: Bearer YOUR_API_KEY"
```

Rule summary:

- Time parameters are ISO-8601; when a from/to pair is used, the span must not exceed **7 days**.
- `from` / `to` = business occurrence time; `received_from` / `received_to` = platform ingest time.
- List items include a **`payload_json` string** (textual form of the business JSON) and **do not** include capture photos.
- Hot events are retained per tenant configuration: default about **7 days**, configurable roughly **3–14 days** (per contract / platform config); the clock is **`received_at`**. After expiry, `GET /events/{id}` may become not-found. Persist promptly.
- The "each time-window pair max 7 days" rule on list queries is a **query-window** limit, separate from on-disk retention days.

List item shape (illustrative; field names match production):

```
{
  "event_id": "evt_a1b2c3d4e5f6478990abcdef12345678",
  "event_type": "punch.recorded",
  "occurred_at": "2026-10-29T14:30:00",
  "received_at": "2026-10-29T14:30:01.123+00:00",
  "connect_device_id": 12,
  "sn": "TEST-SN-001",
  "payload_json": "{\"event_id\":\"evt_a1b2…\",\"sn\":\"TEST-SN-001\",\"user_id\":\"1001\",\"occurred_at\":1761
}
```

Parse `payload_json` once with `JSON.parse`. Note the outer field is **`event_type`**, not `type`. The HTTP body pushed by Webhook / MQTT is the **business JSON obtained by parsing `payload_json`** (not the full list-row wrapper).

Example: Webhook failed overnight and missed 20 events. In the morning, page with `received_from` covering last night's window and upsert into your business DB by `event_id` to catch up.

5.2 Single event and capture photo

```
curl -s "https://api.bnetlink.com/connect/v1/events/{eventId}" \
  -H "Authorization: Bearer YOUR_API_KEY"

curl -s "https://api.bnetlink.com/connect/v1/events/{eventId}/photo" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

The photo endpoint is available only when a punch has a stored capture (commonly `photo_status=stored`). The response is **JSON**, not a raw image byte stream, for example:

```
{
  "event_id": "evt_…",
  "content_type": "image/jpeg",
  "byte_length": 12345,
  "sha256": "…",
  "data_base64": "/9j/4AAQ…"
}
```

Decode `data_base64` and write it to disk. No photo → `404` `photo_not_found`; event missing → `404` `event_not_found`. List / detail responses themselves do not carry image bytes.

5.3 Common event types

event_type	Meaning
`punch.recorded`	Punch
`enroll.data.received`	Enrollment / credential-change summary
`command.completed` / `command.failed`	Command terminal state
`door.state.changed`	Door-sensor state change (device-initiated report)
`door.remote.recorded`	Remote door action (`mode` : pulse/hold/close; not a punch)
`device.presence`	Device online/offline edge (`online` : 1/0)

New rows use only the types above. `GET /devices` `online` is `1` when `last_seen_at` is within **10 minutes**, otherwise `0`. `device.presence` is edge-only: an uplink writes `online=1`; a scan about every **5 minutes** writes `online=0`, so the offline event appears about **10–15 minutes** after the last packet. `occurred_at` is the last uplink; `received_at` is when Connect recorded the edge. Legacy names `device.online` / `device.offline` and `door.open.recorded` / `door.close.recorded` still filter and subscribe, but only the matching edge.

5.3.1 Punch `payload_json` fields (after parse)

Field	Type	Description
`event_id`	string	Event id; dedupe primary key
`sn`	string	Device serial number
`user_id`	string	Person id
`occurred_at`	number	Business occurrence time, **millisecond** timestamp
`device_time`	string	Device local time string (commonly `yyyyMMddHHmmss`)
`verify_mode`	string	Verify-mode description (e.g. Face / Card)
`door_mode`	string	Optional. Device door text (e.g. `hand_open` / `open`)
`door_opened`	boolean	**P5 / BS only.** `true` when the device sent a non-blank `doorMode` (lock driven); `false` when missing or blank. Result only, no deny reason. `remote_open` / `remote_close` are not punches
`has_log_photo`	boolean	Whether a capture is declared
`photo_status`	string	`absent` , `stored` , `omitted_too_large` , or `omitted_invalid`

5.4 Latest-event summary (optional)

```
curl -s "https://api.bnetlink.com/connect/v1/sync/status" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Useful for a coarse check of whether new events exist; **not** a watermark cursor and must not replace your own read-progress position.

---

6. Webhook (optional)

After you register an HTTPS callback, Connect POSTs to you when events occur (with signature).

Recommendation: use Webhook only as a **wake-up**; authoritative data should still come from follow-up `GET /events` pulls.

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/webhooks" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"url\":\"https://example.com/hooks/connect\",\"secret\":\"at-least-16-chars!\",\"event_types\":[]}"
```

Others: `GET /webhooks` (list), `PUT /webhooks/{id}`, `POST /webhooks/{id}/test`.

There is **no** separate `GET /webhooks/{id}`: take detail fields from the list result by `id`.

`PUT` may change `url` / `event_types` / `status` (`ACTIVE` | `DISABLED`); when changing `secret`, supply the new value—**omit to keep the existing secret**. An empty `event_types` means subscribe to all types.

Key points:

- Production public URLs must be **HTTPS** and must not point at private / loopback networks.
- `secret` is 16–128 characters, chosen by you; **do not** use the API Key as the Webhook secret.
- List / detail **no longer return** secret plaintext; save it yourself at create time.
- Callback HTTP body = event business JSON (same as parsed `payload_json`), **not** a photo.

Signature verification (required)

Connect uses HMAC-SHA256:

```
signing string = timestamp + "." + raw_body
X-Epordo-Signature = hex_lower( HMAC_SHA256(secret, signing string) )
```

Request headers also include: `X-Epordo-Timestamp`, `X-Epordo-Event-Id`, `X-Epordo-Event-Type`.

Receivers must verify over the **raw body bytes** and return any **2xx** for success; failures are retried and may eventually become `DEAD`.

Example (pseudocode):

```
expected = hmac_sha256_hex(secret, header["X-Epordo-Timestamp"] + "." + rawBody)
if expected != header["X-Epordo-Signature"]: reject
else: upsert_by_event_id(JSON.parse(rawBody)); return 200
```

Delivery queue (troubleshooting)

Method	Path	Description
GET	`/webhooks/outbox`	Queue rows: `PENDING` / `FAILED` / `SENT` / `DEAD`
GET	`/webhooks/deliveries`	Per-attempt audit (status code, latency, error summary)
POST	`/webhooks/outbox/{id}/replay`	Only `FAILED` / `DEAD` are replayable

Example: On Webhook receipt, only note "new data available", then immediately call `GET /events` with your last saved time window; even if one callback is lost, business data is not lost.

6.1 MQTT (optional; can run alongside Webhook)

Connect can publish the **same** event JSON as Webhook to a fixed Topic. It may run **in parallel** with Webhook; dedupe by `event_id`.

Item	Description
Topic	`epc/{tenant_id}/events/{event_type}` (platform-fixed)
Payload	Same business JSON as events / Webhook
QoS	0 or 1 (default 1)
Broker	Provided by the platform; issue a subscription account on the portal MQTT page for username / password
Connection	Servers commonly use TCP; browsers commonly use WSS (exact addresses shown in the portal)

```
# Register which events to push (one record per tenant)
curl -s -X POST "https://api.bnetlink.com/connect/v1/mqtt" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"event_types\":[],\"qos\":1}"

# Issue a subscription account (password shown only once)
curl -s -X POST "https://api.bnetlink.com/connect/v1/mqtt/credentials" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Notes: the subscription account is **not** an API Key and **not** the Webhook secret. Device traffic still goes through the device server and **does not** become device-direct MQTT.

---

7. Canonical commands

7.1 Create and query

```
# Issue (time sync example)
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/set-time" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{}"

# Query status
curl -s "https://api.bnetlink.com/connect/v1/commands/{command_id}" \
  -H "Authorization: Bearer YOUR_API_KEY"

# Only QUEUED can be cancelled
curl -s -X POST "https://api.bnetlink.com/connect/v1/commands/{command_id}/cancel" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Common fields on successful create:

```
{
  "command_id": "cmd_a1b2c3d4e5f6",
  "sn": "TEST-SN-001",
  "command": "set-time",
  "state": "QUEUED"
}
```

Field	Description
`command_id`	e.g. `cmd_` + hex
`state`	`QUEUED` → `SENT` → `SUCCEEDED` / `FAILED` / `CANCELLED`
`result`	May include a result object at terminal state (command-dependent)

- Optional header `X-Idempotency-Key` (≤128): same Key + same body → return the original command; same Key + different body → `409` `idempotency_conflict`.
- When the device is online: ordinary commands fail after about 3 minutes with no progress; multi-packet read-back inter-packet idle about 10 minutes. At most one `SENT` command per device at a time.
- When the device is offline, commands stay in `QUEUED` until online (still subject to timeout and quota).

Example: Network jitter caused you to retry unlock twice. Using the same `X-Idempotency-Key` and the same JSON both times enqueues only once, so the door does not open twice.

7.2 Command catalog

Canonical name	Path suffix	Purpose
`set-time`	`set-time`	Time sync
`query-status`	`query-status`	Query device status
`unlock`	`unlock`	Remote unlock / hold open / close
`set-user`	`set-user`	Enroll one person (optional templates)
`delete-user`	`delete-user`	Delete one or more persons
`list-user-ids`	`list-user-ids`	Read back person-id list
`get-user`	`get-user`	Read back persons and persist platform enrollment
`set-access-schedule`	`set-access-schedule`	Access time windows
`clear-managers`	`clear-managers`	Clear manager flags (does not delete people)
`pull-logs`	`pull-logs`	Backfill punches by time window

Currently **not provided**: remote enrollment UI, ring settings, etc. Do not integrate against them.

7.3 `set-time`

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/set-time" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"time\":\"20261108120000\"}"
```

`time` is optional, format `yyyyMMddHHmmss`; if omitted, the server uses its local clock at enqueue time.

7.4 `query-status`

No body. On success, `result.model` is this packet's Header `dev_model` (both P5 and P7 send `P5`), not the device's sales `model_code`. Firmware, capacity, and other fields are included (unsupported items may be `null`).

7.5 `unlock`

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/unlock" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"mode\":\"pulse\"}"
```

mode	Meaning
`pulse` (default)	Pulse open, then close after device delay
`hold`	Hold open
`close`	Close

`SUCCEEDED` only means the device acknowledged execution; it **does not** mean the door sensor is open.

7.6 `set-user` (one person · one device)

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/set-user" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"user_id\":\"1001\",\"name\":\"Zhang San\",\"role\":\"user\",\"card\":\"123456\"}"
```

Full field example:

```
{
  "user_id": "1001",
  "name": "Zhang San",
  "role": "user",
  "card": "123456",
  "password": "666",
  "valid_from": "20260101",
  "valid_to": "20261231",
  "replace": false,
  "face": "<Base64>",
  "photo": "<Base64>",
  "fingerprint": ["<Base64>"],
  "palm": ["<Base64>"]
}
```

- `user_id` is required; `fingerprint` / `palm` have at most 10 slots; an **empty array clears that modality**.
- Connect **does not** convert template formats.
- Use `set-access-schedule` for access windows; do not stuff them into this command.

7.7 `delete-user` (multi-person · one device)

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/delete-user" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"user_ids\":[\"1001\",\"1002\"]}"
```

- One command may delete multiple persons (max 100); do not send both `user_id` and `user_ids`.

- There is **no** "clear all persons on the device" operation.

7.8 `list-user-ids` / 7.9 `get-user`

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/list-user-ids" \
  -H "Authorization: Bearer YOUR_API_KEY" -H "Content-Type: application/json" -d "{}"

curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/get-user" \
  -H "Authorization: Bearer YOUR_API_KEY" -H "Content-Type: application/json" \
  -d "{\"user_ids\":[\"1001\"]}"
```

- `get-user` may also use `{"all":true}` (silent persist to avoid event storms).
- After persisting specified ids you usually get `enroll.data.received`.

7.10 `set-access-schedule`

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/devices/TEST-SN-001/commands/set-access-schedule" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"time_groups\":{\"mon\":[\"09001200\",\"14001800\"],\"tue\":[\"09001800\"]},\"user_ids\":[\"1001\",\"100
```

Field	Description
`time_groups`	Required. Keys `sun` … `sat`; values are `HHmmHHmm` string arrays, **1–6 segments per day**
`user_ids`	Optional; omit / empty = all persons on the device

Schedules exist only on the device. To verify, issue `get-user` again and inspect `access_schedules`.

Example: Monday 09:00–12:00 and 14:00–18:00 → `"mon": ["09001200", "14001800"]`.

7.11 `clear-managers` / `pull-logs`

- `clear-managers`: no body; clears manager flags.
- `pull-logs`: `{"from":"…","to":"…"}` (8 or 14 digits; span ≤31 days).

---

8. Fan-out by serial-number list

```
POST /commands/fan-out/set-user
POST /commands/fan-out/delete-user
POST /commands/fan-out/set-access-schedule
```

Common field: `sns` (non-empty string array).

8.1 Principles

On the same device, pack the same operation into **one** command when possible (consistent with device capability), rather than "one HTTP call per person".

API	Recommended shape	Queue behavior
Set users	`users:[{user_id,…}]`	**One command per SN**
Delete users	`user_ids:[…]`	**One command per SN**

Access schedule	Shared `time_groups` + optional `user_ids`	**One command per SN**

8.2 Limits (defaults)

Item	Default
Distinct `sn` count	60 (may drop to 30 for large templates)
Person id count	100
Resulting queued commands	200

Over limit → `400` `fan_out_too_large`.

8.3 Multi-device set users (example)

Prefer `users[]` (each person may have independent name / card / templates); **one command is queued per SN**:

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/commands/fan-out/set-user" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"sns\":[\"SN-A\",\"SN-B\"],\"users\":[{\"user_id\":\"1001\",\"name\":\"Zhang San\",\"card\":\"123\"},{\"
```

Large face templates may exceed a single-device buffer; page or shrink templates before sending.

8.4 Delete many persons on many devices (example)

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/commands/fan-out/delete-user" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d "{\"sns\":[\"SN-A\",\"SN-B\"],\"user_ids\":[\"1001\",\"1002\"]}"
```

Response shape:

```
{
  "command": "delete-user",
  "queued": [
    { "sn": "SN-A", "user_ids": ["1001", "1002"], "command_id": "cmd_…" }
  ],
  "skipped": [
    { "sn": "SN-X", "reason": "device_disabled" }
  ]
}
```

Poll each `command_id` separately. Common `skipped.reason` values: `device_not_found`, `device_disabled`, quota-related.

8.5 Same access schedule on many devices (example)

```
{
  "sns": ["SN-001", "SN-002"],
  "time_groups": {
    "mon": ["09001200", "14001800"],
    "tue": ["09001800"]
  },
  "user_ids": ["1001", "1002", "1003"]
}
```

---

9. Device groups and Scoped Keys

Path prefix: `/device-groups`. Creating / renaming / enabling-disabling groups requires an account-wide (ACCOUNT) Key.

There is **no** `DELETE /device-groups/{id}`: to disable, `PUT` with `status` set to `DISABLED`.

Method	Path	Description
GET	`/device-groups`	List
POST	`/device-groups`	Create: `{"code":"store-a","name":"Store A"}`
GET	`/device-groups/{id}`	Detail + member devices
PUT	`/device-groups/{id}`	Change `name` / `status` (`ACTIVE` \ `DISABLED`; `code` is immutable)
PUT	`/device-groups/{id}/devices/{sn}`	Add an already-claimed device
DELETE	`/device-groups/{id}/devices/{sn}`	Remove a member
POST	`/device-groups/{id}/commands/set-time`	Group batch time sync
POST	`/device-groups/{id}/commands/query-status`	Group batch status query
POST	`/device-groups/{id}/commands/unlock`	Group batch unlock (`mode` same as single-device)

```
curl -s -X POST "https://api.bnetlink.com/connect/v1/device-groups" \
  -H "Authorization: Bearer YOUR_API_KEY" -H "Content-Type: application/json" \
  -d "{\"code\":\"store-a\",\"name\":\"Store A\"}"

curl -s -X PUT "https://api.bnetlink.com/connect/v1/device-groups/1/devices/TEST-SN-001" \
  -H "Authorization: Bearer YOUR_API_KEY"

curl -s -X POST "https://api.bnetlink.com/connect/v1/device-groups/1/commands/query-status" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Group batch responses are always **200** (partial success allowed), shaped like fan-out: `queued[]` / `skipped[]`.

Notes:

- Users / delete / access schedules have **no** group-level batch; use §8 fan-out (explicit `sns`).
- You may issue API Keys with `scope_type=GROUPS` for a group (see §2.5).
- Group Keys **cannot** log into the portal, and cannot claim / unbind / manage Webhooks, MQTT, credentials, or groups themselves.
- When group status is `DISABLED`, group-level commands may return `409` `group_disabled`.

---

10. Rate limits and daily quotas (summary)

- **Per-minute**: request rate per Key; over limit → `429` `rate_limited` (response may include `Retry-After`).
- **Daily quota (UTC day)**: "expensive" commands such as unlock, full read-back, log pull, set/delete users, schedules; abnormal usage may auto-disable the Key (`quota_exceeded_key_revoked`).
- Idempotent replays usually **do not** consume quota again. Exact numbers follow contract / platform configuration.

---

11. Recommended integration flow

11.1 Minimal loop (punches)

1. Obtain an API Key; confirm with `GET /me`.
2. Claim with `POST /devices`; configure the device-server URL on the terminal and bring it online.
3. (Optional) Register Webhook or MQTT.
4. Periodically or after wake-up, pull again with `GET /events`; store your own progress.
5. Fetch captures with `GET /events/{id}/photo` when needed.

11.2 User provisioning

1. Maintain persons and templates on the business side.
2. Single device: `set-user`; multi-device: `fan-out/set-user`.
3. Same access rules: `set-access-schedule` or fan-out with shared `time_groups`.
4. Verify with `list-user-ids` / `get-user`.

11.3 Same-series device fleets

- Sales models stay distinct (P5 and P7).
- Do not run full `get-user` on every unit.
- Treat ERP or one representative device as authoritative; reconcile other units with `list-user-ids` and fill gaps.
- Compatible list `GET /devices/{sn}/users?scope=compatible` crosses devices only when the manufacturer assigned the same template family and the firmware is the same and non-empty. Header `dev_model` does not change the sales model. An empty firmware does not cross devices.

Appendix A. Common error codes

code	Typical scenario
`missing_bearer` / `empty_key` / `invalid_key`	Authentication failure
`missing_sn`	Claim without serial
`device_not_in_stock`	Serial not in vendor stock
`device_not_for_tenant`	Stock not assigned to this account
`device_not_found`	Device missing or out of scope
`device_already_registered` / `sn_already_registered`	Duplicate registration
`device_disabled`	Device disabled
`invalid_set_user` / `invalid_delete_user` / `invalid_access_schedule`	Invalid command body
`invalid_fan_out` / `fan_out_too_large`	Bad batch params or over limit
`command_not_found` / `command_not_cancelable`	Command query / cancel
`idempotency_conflict`	Idempotency-key conflict
`rate_limited` / `quota_exceeded` / `quota_exceeded_key_revoked`	Rate limit / quota
`invalid_webhook_url` / `invalid_webhook_secret`	Webhook parameters
`forbidden_scope`	Group Key called an account-level API

`last_active_api_key`	Attempted to disable the last active Key
`api_key_not_found`	Credential id not found
`mqtt_already_exists`	Tenant already registered MQTT outbound; do not POST again
`outbox_not_replayable`	Delivery-queue state not replayable
`group_not_found` / `group_disabled`	Group missing or disabled

Appendix B. Credential kinds

kind	Slots	Description
`photo`	0	Visible-light photo
`face`	0	Face template
`fingerprint`	0–9	Fingerprint
`palm`	0–9	Palm print / palm vein
`card`	0	Card number (UTF-8)
`password`	0	Device password (UTF-8)

A single decoded blob over about **1MB** is rejected. The platform does not convert templates across firmware.

Appendix C. Documentation and support

Material	Description
This document / PDF	Public English API guide (PDF downloadable from portal Help)
Portal	Browser claim, try commands, issue MQTT subscription accounts, etc.
ApiPost / Postman	Import collection delivered with the project (technical attachment)
Issue feedback	Commercial / technical support channels

Appendix D. API quick reference

Category	Methods and paths			
Health / identity	`GET /health` · `GET /me` · `GET	POST /credentials · PUT /credentials/{id}`		
Devices	`POST	GET /devices · GET	DELETE /devices/{sn} · POST .../disable	enable · GET .../users...`

Events	`GET /events` · `GET /events/{id}` · `GET /events/{id}/photo` · `GET /sync/status`				
Webhook	`GET	POST /webhooks · PUT /webhooks/{id} · POST .../test · GET .../outbox	deliveries · POST .../outbox/{id}/replay` (no separate GET by id)		
MQTT	`GET	POST /mqtt · PUT	DELETE /mqtt/{id} · POST .../test · GET	POST	DELETE /mqtt/credentials` · outbox/deliveries/replay
Commands	`POST /devices/{sn}/commands/{name}` · `GET /commands/{id}` · `POST /commands/{id}/cancel`				
Batch	`POST /commands/fan-out/set-user	delete-user	set-access-schedule`		
Groups	`GET	POST /device-groups · GET	PUT /device-groups/{id} · `member PUT/DELETE` · `group set-time` / `query-status` / `unlock` (no DELETE of the group itself)		

Canonical command `{name}` values: `set-time`, `query-status`, `unlock`, `set-user`, `delete-user`, `list-user-ids`, `get-user`, `set-access-schedule`, `clear-managers`, `pull-logs`.

End of document · EPORDO Connect API Guide v1.1.5