



myeztoll — Partner API: Quick Start with Examples

API	https://owner.myeztoll.com/api/partner/v1
Reference (public, try-it-out)	https://owner.myeztoll.com/api/partner/v1/index
Guide version	2026-09-30 · API v1.4
For	partners' developers — the shortest path from a key to working calls

Reference (public, try-it-out): <https://owner.myeztoll.com/api/partner/v1/index>

What this document is

Five short sections, one per video in the **Partner API** playlist. Each shows the calls you actually need, with a request and a real-shaped response. Every other function — 54 in all, each with its own example — is in the reference above, and in the catalog the API serves itself (`GET /api/partner/v1` , no key needed).

1. Your key and the first call

Every call carries a key in the `X-API-Key` header.

Credential	Looks like	Acts for
Owner key	<code>aegispk-...</code>	One owner — the account it was issued for
Integration token	<code>aegispi-...</code>	Many owners — each call names the owner in <code>X-Owner-Id</code>

- Keys are issued by us. An owner sees their own key, read-only, in their portal.
- An **integration** acts only for owners who connected it from their own portal. `GET /owners` lists them; an owner who has not connected you (or disconnected) answers `401` .
- **Test first on the sandbox**, <https://dev-owner.myeztoll.com/api/partner/v1> , with a key of its own.

The envelope. Every response has the same shape — except a licence-image or evidence-file download, which returns the raw file:

```
{ "success": true, "data": { ... }, "error": null }
```

When `success` is `false` , `error` says why.

Conventions

- **Dates** are UTC ISO-8601 (`2026-07-31T14:05:00Z`). A range includes `from` and excludes `to` .
- **Ids**: wherever an id is accepted you may pass **ours or your own** `externalId` . Create calls accept `externalId` ; responses echo it back.
- **Paging**: every list is a page — `data = { items, page, pageSize, total, hasMore } . ?`
`page=1&pageSize=100` (default 100, max 1000). Keep asking while `hasMore` is `true` .

Set up once (the sandbox: `BASE=https://dev-owner.myeztoll.com/api/partner/v1` with its own key):

```
KEY=aegispk-... # or your aegispi-... integration token
BASE=https://owner.myeztoll.com/api/partner/v1
```

First call — who is this key?

```
curl -H "X-API-Key: $KEY" "$BASE/me"
```

```
{
  "success": true,
  "data": { "kind": "integration", "integrationId": "...", "name": "Your Platform",
            "connectedOwners": 12, "actingOwnerId": null },
  "error": null
}
```

2. Cars, drivers and bookings — with your own ids

Create a car (use your own id in `externalId`; from then on it works anywhere ours does):

```
curl -X POST -H "X-API-Key: $KEY" -H "Content-Type: application/json" \
  $BASE/cars \
  -d '{"externalId":"CAR-001", "title":"Camry", "carTypeId":"...",
      "stateId":"NJ", "city":"Newark", "zipCode":"07102",
      "phoneNumber":"+12015550100", "carPlateNumber":"ABC1234",
      "carBrand":"Toyota", "color":"Black", "interiorColor":"Black",
      "year":2022}'
```

```
{ "success": true,
  "data": { "id": "5d73...", "externalId": "CAR-001",
            "title": "Camry", "plateNumber": "ABC1234" },
  "error": null }
```

`carTypeId` comes from `GET /car-types`; `stateId` from `GET /states`.

Attach the toll transponder so the car's tolls are matched to it (`tollProviderId` from `GET /toll-providers`):

```
curl -X POST -H "X-API-Key: $KEY" -H "Content-Type: application/json" \
  $BASE/cars/CAR-001/transponders \
  -d '{"tollProviderId":"...", "transponderNumber":"01234567890"}'
```

Add a driver (multipart; licence photos optional). Sending the same e-mail again updates the driver instead of adding a second one:

```
curl -X POST -H "X-Api-Key: $KEY" \
  -F firstName=John -F lastName=Doe -F email=john@acme.com -F externalId=DRV-001 \
  -F frontSideImage=@front.jpg \
  $BASE/drivers
```

Record a booking — a car, a driver and a time window, all by your own ids. It is only a record: creating it charges nobody.

```
curl -X POST -H "X-Api-Key: $KEY" -H "Content-Type: application/json" \
  $BASE/bookings \
  -d '{"externalId":"BK-001","carId":"CAR-001","renterId":"DRV-001",
    "dateFrom":"2026-08-01T00:00:00Z","dateTo":"2026-08-04T00:00:00Z"}'
```

```
{ "success": true,
  "data": { "id": "b1...", "externalId": "BK-001",
            "carId": "5d73...", "carExternalId": "CAR-001",
            "renterId": "9c1f...", "renterExternalId": "DRV-001",
            "status": 6, "statusName": "Approved" },
  "error": null }
```

Can this driver be charged? `hasCard` = a card is on file; `isValid` = it can really be charged (not expired, not a wallet we cannot charge, not suppressed after failures).

```
curl -H "X-Api-Key: $KEY" "$BASE/drivers/DRV-001/card"
```

If not, `POST /bookings/BK-001/card-link` returns `cardRegistrationUrl` — a page where the driver adds a card.

Bookings are optional. Without them you can still read every toll by car (section 3).

3. Tolls and fines

Every priced toll in a period, from the toll agencies and from GPS:

```
curl -H "X-Api-Key: $KEY" \
  "$BASE/tolls?from=2026-07-01T00:00:00Z&to=2026-08-01T00:00:00Z&page=1&pageSize=100"
```

One row (trimmed):

```
{
  "source": "agency",
  "externalCarId": "CAR-001",
  "plateNumber": "ABC1234", "plateState": "NJ",
  "occurredAtUtc": "2026-07-26T04:51:45Z",
  "agency": "GSP", "location": "KEY - Keyport",
  "agencyAmount": 0.79,
  "serviceFee": 5,
  "currency": "USD",
  "bookingSource": "turo", "bookingSourceReservationId": "41234567",
  "renterCharge": { "status": "charged", "amount": 6.45, "paidAmount": 18.2,
    "chargedAtUtc": "2026-07-28T09:12:03Z", "failureReason": null }
}
```

Field	Meaning
<code>source</code>	<code>agency</code> (the toll agency's record) or <code>gps</code> (priced from the car's track)
<code>agencyAmount</code>	What the road really cost — bill your renter this
<code>serviceFee</code>	Our separate fee (nullable). Never added into the road cost for you
<code>bookingSource</code>	Where the booking came from (<code>turo</code> , <code>hqrental</code> , <code>own</code> , ...), with its reservation id
<code>renterCharge.status</code>	Did the renter pay it back: <code>not_billed</code> , <code>estimate</code> , <code>pending</code> , <code>charged</code> , <code>declined</code> , <code>invoiced</code> , <code>reimbursed_by_platform</code> , <code>paid_cash</code> , <code>paid_manually</code> , <code>owner_absorbed</code>
<code>renterCharge.paymentId</code>	Tolls collected in one card charge share it (so <code>paidAmount</code> can exceed <code>amount</code>)

Other ways in

- One car, any period, no bookings needed: `GET /cars/CAR-001/tolls?from=&to=` (agency tolls).
- One rental: `GET /bookings/BK-001/tolls` — GPS tolls carry an id starting with `gps:` .
- Fines: `GET /violations?from=&to=` — `amount` is the fine, `serviceFee` ours, `isPaid` whether it is settled.

Read every page (Node.js 18+):

```
const BASE = 'https://owner.myeztoll.com/api/partner/v1';

async function allTolls(key, from, to) {
  const rows = [];
  for (let page = 1; ; page++) {
    const url = `${BASE}/tolls?from=${from}&to=${to}&page=${page}&pageSize=1000`;
    const res = await fetch(url, { headers: { 'X-Api-Key': key } });
    const body = await res.json(); // { success, data, error }
    if (!body.success) throw new Error(body.error);
    rows.push(...body.data.items);
    if (!body.data.hasMore) return rows; // keep asking while hasMore
  }
}
```

4. GPS and webhooks

Where every tracked car is now — GET `/gps/positions`. Each item carries `lat`, `lon`, `speedMph`, `ignition`, and freshness: `online` and `ageSeconds`.

How far a car drove — GET `/cars/CAR-001/gps/mileage?from=&to=` → `distanceMiles`, `distanceKm`, `positionCount`. For one rental: GET `/bookings/BK-001/gps/mileage`. Raw positions are kept 30 days; priced GPS tolls are permanent (`/tolls?source=gps`).

Be told instead of asking — register a webhook:

```
curl -X POST -H "X-Api-Key: $KEY" -H "Content-Type: application/json" \
  -d '{"url":"https://example.com/hooks/myeztoll"}' \
  "$BASE/webhooks"
```

```
{ "success": true,
  "data": { "id": "...", "url": "https://example.com/hooks/myeztoll",
            "events": ["*"], "active": true, "secret": "whsec_..." },
  "error": null }
```

- `"*"` = every event: `toll.created`, `toll_charge.updated`, `violation.created`, `violation.charge_updated`, `owner.connected`, `owner.disconnected`, `dispute.evidence_ready`, `dispute.evidence_failed`, `dispute.deadline_approaching`.
- The **secret is shown once** — store it. POST `/webhooks/{id}/test` sends a test event; GET `/webhooks/{id}/deliveries` is the delivery log.

Check every delivery. Header `X-MyEzToll-Signature: t=<unix seconds>,v1=<hex>`, where `v1` = HMAC-SHA256(secret, "<t>.<raw body>"):

```
const crypto = require('crypto');

function verify(rawBody, header, secret) {
  const parts = Object.fromEntries(header.split(',').map((p) => p.split('=')));
  const signed = `${parts.t}.${rawBody}`; // "t.body", the body as received
  const expected = crypto.createHmac('sha256', secret).update(signed, 'utf8').digest('hex');
  const fresh = Math.abs(Date.now() / 1000 - Number(parts.t)) < 300; // reject replays
  return fresh && String(parts.v1).length === expected.length
    && crypto.timingSafeEqual(Buffer.from(expected), Buffer.from(parts.v1));
}
```

Also sent: `X-MyEzToll-Event` (the event type) and `X-MyEzToll-Event-Id` — de-duplicate on it; a delivery is retried with back-off for up to about 1.5 days.

5. Your charges and disputes

For charges **you** make on **your own Stripe account**. We never call your Stripe account.

1. Report the charge — which tolls, fines and fees it covered:

```
curl -X POST -H "X-API-Key: $KEY" -H "Content-Type: application/json" \
  -d '{"chargeId": "ch_3Qabc123", "amount": 25.64,
    "chargedAtUtc": "2026-09-20T14:02:11Z", "bookingId": "BK-001",
    "items": [{"type": "toll", "id": "34333995590"},
              {"type": "fee", "amount": 5, "label": "Service fee"}]}' \
  "$BASE/charges"
```

We match each item to our records and check the sum: if the items do not add up to the charge, `warnings` carries `sum_mismatch` ("Report your own fees as items of type fee").

2. When Stripe opens a dispute on it, report it:

```
curl -X POST -H "X-API-Key: $KEY" -H "Content-Type: application/json" \
  -d '{"stripeDisputeId": "du_1UDk3z", "chargeId": "ch_3Qabc123",
    "reason": "product_not_received", "amount": 25.64,
    "evidenceDueBy": "2026-10-05T23:59:59Z"}' \
  "$BASE/disputes"
```

→ `"evidenceStatus": "collecting"`. When the package is ready we send `dispute.evidence_ready`.

3. Read the package — `GET /disputes/du_1UDk3z`. `evidence` is keyed by **Stripe's own evidence field names**: pass the text fields through as they are; for each `{ "fileId": ... }` download the file (`GET /disputes/du_1UDk3z/files/{fileId}`), upload it to your Stripe account (purpose `dispute_evidence`) and

put Stripe's `file_...` id in its place. `recommendation` says whether to contest or accept. Add your own documents with `POST /disputes/{id}/documents`.

4. Submit on your side, then tell us:

```
curl -X PATCH -H "X-API-Key: $KEY" -H "Content-Type: application/json" \
  -d '{"status": "submitted"}' "$BASE/disputes/du_1UDk3z"
```

...and later `won` or `lost`.

Where to go next

- **Reference with try-it-out:** <https://owner.myeztoll.com/api/partner/v1/index>
- **OpenAPI document:** <https://owner.myeztoll.com/api/partner/v1/openapi.json>
- **The catalog, every function with an example:** `GET https://owner.myeztoll.com/api/partner/v1`
- **Videos:** the **Partner API** playlist (episodes 41–45).