prism

Webhook Integration Guide

How to receive Prism Connect events, verify their signature with your signing secret, and build a handler that stays correct under retries.

Prism Connect · `api.prism.horse/connect/v1` · This document describes the live contract as implemented.

1. How it works

Prism pushes events to an HTTPS endpoint you control. You configure that endpoint once in the Prism portal, prove you own it, and then receive a signed JSON POST every time something happens on the account.

- **Register** — in Prism, open *Integration* → *Webhook*, enter your HTTPS URL and tick the modules you want. Copy the signing secret; it is shown once.
- **Verify** — Prism immediately POSTs a `webhook.verification` event. Answer **HTTP 200** and your endpoint becomes ACTIVE.
- **Receive** — every subscribed event is POSTed to your URL, signed, and retried on failure.

> **Until the handshake passes, no product events are sent.** Until the handshake passes, your endpoint stays PENDING_VERIFICATION and receives NO product events — only the verification request itself. This is the single most common reason a new integration appears silent. 5 verification attempts per hour per webhook.

2. Endpoint requirements

- HTTPS only. Plain HTTP is rejected at registration.
- Port 443 or 8443.
- A publicly resolvable host. Private, loopback and link-local addresses are rejected, and the host is re-checked at delivery time.
- No credentials embedded in the URL.

3. What we send

Every delivery is a `POST` with a JSON body and these headers:

Header	Example	Meaning
`X-Prism-Event`	`horse.report.group.created`	Event type (wire name). Switch on this — never on the module.
`X-Prism-Event-Id`	`8e9e30d5-9934-406a-842c-a38e06ce4443`	Identifies the event. Stable across retries of the same event.
`X-Prism-Delivery-Id`	`0ed43984-8494-4536-b3f0-aa227f435af0`	Identifies this delivery attempt chain. Use it as your idempotency key.
`X-Prism-Delivery-Attempt`	`1`	1 on the first attempt, incrementing on each retry. Absent on the verification request.
`X-Prism-Signature-Timestamp`	`1785421852`	Unix time in SECONDS at signing. Part of the signed string.
`X-Prism-Signature`	`sha256=6f3a…`	HMAC-SHA256 of timestamp + "." + raw body, lowercase hex.
`Content-Type`	`application/json`	Always JSON.
`User-Agent`	`Prism-Webhook/1.0`	Prism-Webhook-Verify/1.0 for the verification request.

The envelope

Field	Type	Notes
`deliveryId`	`string`	Stable across retries — deduplicate on this.
`eventId`	`string`	Identifies the event itself.
`eventType`	`string`	Wire name, e.g. "task.trackwork.created".
`occurredAt`	`string`	ISO 8601 UTC instant, e.g. "2026-07-30T04:00:00Z".
`accountId`	`number`	The Prism account the event belongs to.
`data`	`object`	Event payload. Its shape is determined by eventType.

```
{
  "deliveryId": "0ed43984-8494-4536-b3f0-aa227f435af0",
  "eventId": "8e9e30d5-9934-406a-842c-a38e06ce4443",
  "eventType": "horse.report.group.created",
  "occurredAt": "2026-07-30T04:00:00Z",
  "accountId": 1234,
```

```
  "data": {
    "accountId": 1234,
    "id": 5678,
    "type": "GROUP_UPDATE",
    "title": "Weekly stable update",
    "status": "PUBLISHED",
    "publishDate": "2026-07-30T04:00:00Z",
    "reportCount": 2,
    "reports": [
      {
        "id": 55101,
        "horseId": 42,
        "horseName": "Winx"
      },
      {
        "id": 55102,
        "horseId": 43,
        "horseName": "Black Caviar"
      }
    ]
  }
}
```

4. Verifying the signature

HMAC-SHA256 over `<X-Prism-Signature-Timestamp> + "." + <raw request body>`, sent as `sha256=<lowercase hex digest>` in `X-Prism-Signature`.

> **Sign the raw body.** Sign the RAW body bytes exactly as received. Re-serialising the parsed JSON changes whitespace and key order, and the signature will not match.

The key is your signing secret, used as raw UTF-8 bytes — do not base64-decode or hash it first.

Node / TypeScript

```
scripts/generate-webhook-pdf.mjs`
 * reads this file and extracts the template literal, so the two can never drift.
 *
 * The code itself is deliberately dependency-light: Express + Node's built-in `crypto`, nothing else.
 */
export const VERIFY_EXAMPLE_TS = `import crypto from 'node:crypto';
import express from 'express';

const app = express();

const SIGNING_SECRET = process.env.PRISM_WEBHOOK_SECRET!; // whsec_… from Integration → Webhook
const TOLERANCE_SECONDS = 5 * 60;                         // reject very old / future timestamps

/**
 * IMPORTANT: express.raw() — the signature covers the RAW body bytes.
 * express.json() would give you a parsed object, and re-serialising it changes
 * whitespace and key order, so the signature would never match.
 */
app.post('/webhooks/prism', express.raw({ type: 'application/json' }), (req, res) => {
  const signatureHeader = req.header('X-Prism-Signature');        // "sha256=<hex>"
  const timestamp = req.header('X-Prism-Signature-Timestamp');    // unix SECONDS
  const rawBody = req.body as Buffer;

  if (!signatureHeader || !timestamp) {
    return res.status(400).send('missing signature headers');
  }

  // 1. Reject stale timestamps — this is what stops a captured request being replayed.
  const ageSeconds = Math.abs(Math.floor(Date.now() / 1000) - Number(timestamp));

  if (!Number.isFinite(ageSeconds) || ageSeconds > TOLERANCE_SECONDS) {
    return res.status(400).send('stale timestamp');
  }

  // 2. Recompute: HMAC-SHA256 over "<timestamp>." + rawBody, keyed with the secret.
  const expected = crypto
    .createHmac('sha256', SIGNING_SECRET)
    .update(timestamp + '.')
    .update(rawBody)
    .digest('hex');

  const received = signatureHeader.replace(/^sha256=/, '');

  // 3. Constant-time compare. timingSafeEqual throws on length mismatch, so guard first.
  const a = Buffer.from(expected, 'utf8');
  const b = Buffer.from(received, 'utf8');
```

```
  const valid = a.length === b.length && crypto.timingSafeEqual(a, b);

  if (!valid) {
    return res.status(401).send('bad signature');
  }

  const event = JSON.parse(rawBody.toString('utf8'));

  // 4. The verification handshake: answer 200 and you are switched to ACTIVE.
  //    Until that happens Prism sends no product events at all.
  if (event.eventType === 'webhook.verification') {
    return res.sendStatus(200);
  }

  // 5. Acknowledge FAST, then work. Prism gives you 10 seconds before it counts a timeout.
  res.sendStatus(200);

  // 6. Delivery is at-least-once: the same deliveryId can arrive again after a retry.
  void handleEvent(event).catch(err => console.error('[prism] handler failed', err));
});

const seenDeliveries = new Set<string>(); // use Redis/DB in production, not memory

async function handleEvent(event: {
  deliveryId: string;
  eventType: string;
  accountId: number;
  data: Record<string, unknown>;
}) {
  if (seenDeliveries.has(event.deliveryId)) return; // idempotency
  seenDeliveries.add(event.deliveryId);

  // Switch on eventType, never on the module: one module can deliver more than one payload shape.
  switch (event.eventType) {
    case 'horse.report.group.created': {
      // A group update: one event carrying every report in the group.
      const { id, reportCount, reports } = event.data as {
        id: number;
        reportCount: number;
        reports: { id: number; horseId: number; horseName: string }[];
      };

      console.log(`group ${id}: ${reportCount} reports`, reports.map(r => r.horseName));
      break;
    }

    case 'task.trackwork.created':
      console.log('new trackwork', event.data);
      break;

    default:
      console.log('unhandled event', event.eventType);
  }
}

app.listen(3000);
```

5. Delivery, retries and idempotency

Rule	Value
Success	Any 2xx is success. Respond quickly — acknowledge first, process asynchronously.
Retried statuses	`500–599, 408, 425 and 429`
Not retried	Any other 4xx is treated as final — we will not retry.
Attempts	5 (initial + 4 retries)
Retry schedule	30 seconds · 2 minutes · 10 minutes · 30 minutes
Response timeout	10 seconds
Connect timeout	5 seconds
Maximum payload	256 KB
Semantics	At least once. A retry re-sends the same deliveryId and eventId, so deduplicate on deliveryId.
Ordering	Delivery order is not guaranteed. Use occurredAt if you need ordering.

Automatic disabling

If at least 10 deliveries in a 60-minute window fail at a rate of 50% or more, The endpoint is set to AUTO_DISABLED and we email the account. Re-enable it from Integration → Webhook with Verify, which re-runs the handshake and resets the counters.

6. Event catalogue

You subscribe per module; each module emits the events below. Always switch on `eventType`.

Module	Events
Trackwork tasks `TASK_TRACKWORK`	`task.trackwork.created` `task.trackwork.updated` `task.trackwork.completed` `task.trackwork.deleted`
Procedure tasks `TASK_PROCEDURE`	`task.procedure.created` `task.procedure.updated` `task.procedure.completed` `task.procedure.deleted`
Transport tasks `TASK_TRANSPORT`	`task.transport.created` `task.transport.updated` `task.transport.completed` `task.transport.deleted`
General tasks `TASK_GENERAL`	`task.general.created` `task.general.updated` `task.general.completed` `task.general.deleted`
Horse notes — one module per note type `HORSE_NOTE_*`	`note.general.*` `note.vet.*` `note.temp.*` `note.weight.*` `note.feed_left.*` `note.scope.*` `note.xray.*` `note.trot_up.*` `note.race_plan.*` `note.blood_profile.*` *Each note type is subscribed separately and emits created, updated and deleted.*
Horse media `HORSE_MEDIA`	`horse.media.created` `horse.media.updated` `horse.media.deleted`
Horse reports and group updates `HORSE_REPORT`	`horse.report.created` `horse.report.updated` `horse.report.deleted` `horse.report.group.created` `horse.report.group.updated` `horse.report.group.deleted` *This module delivers TWO payload shapes: a single report, and a group update carrying a reports[] array. Switch on eventType.*

Module	Events
Stable updates `GENERAL_UPDATE`	`general_update.created` `general_update.updated` `general_update.deleted`
System `—`	`webhook.test` `webhook.verification` *System events bypass module subscriptions. webhook.test is sent from the portal's Send test button.*

7. Troubleshooting

Symptom	Cause	Fix
The endpoint receives nothing at all, and no errors appear anywhere.	The webhook is still PENDING_VERIFICATION. Product events are withheld until the handshake passes.	Open Integration → Webhook and press Verify. Your endpoint must answer the verification request with HTTP 200.
Some event types arrive, others never do.	The missing events belong to a module the webhook is not subscribed to.	Tick the module in Integration → Webhook → Events and save.
Signature never matches.	Almost always the body: the signature covers the raw bytes, and most frameworks hand you a re-serialised object.	Capture the raw body before any JSON parsing — see the express.raw() example.
Deliveries stopped after a period of failures.	Auto-disable tripped: at least 10 deliveries in a 60-minute window with a failure rate of 50% or more.	Fix the endpoint, then press Verify to re-enable it and reset the counters.
The same event arrives more than once.	Expected. Delivery is at least once — a retry re-sends the same deliveryId.	Deduplicate on deliveryId and make your handler idempotent.