LawnPro Software

API & Webhooks

Developer documentation for integrating with LawnPro.

Integration Guide

Step-by-step: from API key to go-live.

API Reference

Every endpoint, field, filter and enumeration.

Webhooks Reference

Delivery format, signature verification, retries, event catalog.

Public REST API v1 · https://secure.lawnprosoftware.com/api/v1
Generated August 3, 2026 · Support 205-369-7052, 7am–7pm CST

LawnPro API & Webhooks — Integration Guide

Start here. This guide walks a working integration end to end: getting a key, making your first call, building a client that survives production, receiving webhooks, and going live. It is instructions, not a catalog.

When you need field-level detail, use the reference docs instead:

Document	Use it for
`README.md`	One-page overview and cheat sheet.
`REFERENCE.md`	Every endpoint, field, filter, and enum.
`WEBHOOKS.md`	Complete webhook spec (envelope, signing, retries, event catalog).
`openapi.yaml`	Machine-readable OpenAPI 3.1 contract — generate a client from this.

- **Base URL:** `https://secure.lawnprosoftware.com/api/v1`
- **Auth:** `Authorization: Bearer lp_live_<prefix>.<secret>`
- **Everything is JSON**, both directions.

> **A note on vocabulary.** LawnPro says **customer** (not client), **property** (not job site), **visit** (not appointment or job), and **estimate** (not quote). The API field names follow that, and so does this guide. Sections marked **LawnPro-specific** describe behavior that won't match your instincts from other platforms — read those even if you skim the rest.

Table of contents

---

1. Before you start

What you need:

- A LawnPro account, and the **account owner's** cooperation — only the owner (`user_role = 1`) can request an API key or manage webhooks. A staff login with full settings access still cannot, deliberately: a key carries the whole company's data.
- Somewhere to keep a secret. The key is shown once and is equivalent to a password for the entire account.
- For webhooks: a public HTTPS URL. Not `localhost` — see Step 8.

What the API can do today:

Resource	Read	Create / update / delete
Customers	yes	yes
Properties	yes	yes
Items & Services	yes	yes
Invoices	yes	**no**
Estimates	yes	**no**
Payments	yes	**no**
Schedule / visits	yes	**no**
Webhook endpoints	yes	yes

LawnPro-specific: invoices, estimates, payments, and the schedule are **read-only in v1**. You can pull them out; you cannot create or change them through the API. If your integration needs to write an invoice, that work does not exist yet — plan around it rather than around a workaround.

Is it turned on? API access is currently a **pilot feature behind a flag**, not a plan entitlement. If **Settings → API Access** isn't visible, the account hasn't been enabled and no amount of correct code will help — contact support first. Webhooks have their own separate flag, so an account can have webhooks without API keys (and enabling API access grants webhooks too).

Step 1 — Get an API key

Keys are **self-service to request, staff-approved to issue**. Three moves:

1. **The owner requests it.** In LawnPro: **Settings → API Access → Request an API key**. Give it a label (e.g. "Acme CRM sync") and say what it's for. Be specific about what the integration does — that text is what staff use to decide the scopes.
2. **LawnPro staff approve it** and set the scopes and rate limit. The request shows as **Awaiting approval** until then, and the owner can **Cancel** it. Only **one open request at a time** per account, so don't queue up several.
3. **The owner generates the key.** Once the row shows **Approved**, the owner clicks **Generate key**.

The full key appears **exactly once**, right then.

Copy it immediately into your secrets manager. Only a public prefix and a bcrypt hash of the secret are stored, so nobody — not support, not the owner — can retrieve it later. Lost key means requesting a new one and revoking the old.

> **The secret is generated in the owner's own browser session at the moment they click Generate key.** LawnPro staff approve the *request*; they never see the resulting secret. If anyone asks you to send them a key, that is not how this works.

Ask for the right scopes

Scopes are `resource:action`, and a read scope never implies write. Ask for the narrowest set that does the job — a reporting dashboard should get `customers:read` and `invoices:read`, nothing more.

`customers:read` · `customers:write` · `properties:read` · `properties:write` · `items:read` · `items:write` · `invoices:read` · `estimates:read` · `payments:read` · `schedule:read` · `webhooks:read` · `webhooks:write`

A call missing its scope fails with `403 insufficient_scope` and names the scope it wanted, so this is quick to diagnose — but changing scopes means a new approval, so get it right at request time.

Step 2 — Make your first call

Prove the key works before you write any code:

```
curl -i "https://secure.lawnprosoftware.com/api/v1/customers?per_page=1" \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>"
```

A `200` with a `data` array means you're connected. Look at the response headers too — they matter later:

```
X-Request-Id: req_2f1c9a7b          <- quote this to support
X-RateLimit-Limit: 120
X-RateLimit-Remaining: 119
```

If it fails, jump to Troubleshooting. The three usual causes are a missing `Bearer ` prefix, a truncated secret, and the account not having the feature enabled.

Never put the key in a browser, mobile app, or anything a customer can view source on. Calls are server-to-server. CORS is restricted to a short allowlist of LawnPro's own origins, so a browser fetch from your domain will be blocked anyway — treat that as a guardrail, not the reason.

Step 3 — Read the envelope correctly

Every response — success and failure alike — has the same three keys:

```
{
  "data":   { "id": 42, "first_name": "Jane", "last_name": "Doe" },
  "meta":   { "request_id": "req_2f1c9a7b" },
  "errors": []
}
```

Lists put the rows in `data` and add pagination to `meta`:

```
{
  "data": [ { "id": 42 }, { "id": 41 } ],
  "meta": {
    "request_id": "req_2f1c9a7b",
    "pagination": { "page": 1, "per_page": 50, "total": 213, "total_pages": 5 }
  },
  "errors": []
}
```

Write your client against this shape once, centrally:

```
def call(method, path, **kw):
    r = requests.request(method, f"{BASE}{path}", headers=HEADERS, timeout=30, **kw)
    body = r.json()
    if body["errors"]:
        raise LawnProError(body["errors"][0]["code"],
                           body["errors"][0]["message"],
                           body["meta"]["request_id"])
    return body["data"], body["meta"]
```

Branch on `errors[0].code`, never on `message`. Codes are a stable contract; message wording is not.

Log `meta.request_id` on every failure. It's the only thing that lets support find your exact call in the server-side audit log.

Step 4 — Handle errors like production will

Four failures will happen to you in the first week. Handle them before launch.

429 — rate limited

Limits are **per key, per minute**, in a fixed 60-second window. Default is **120 requests/minute**; support can raise it per key. You get:

```
Retry-After: 37
```

Sleep that many seconds and retry. Don't guess, don't hammer — and don't ignore `X-RateLimit-Remaining`, which lets you slow down *before* you get blocked.

503 — the limiter itself is unavailable

Rare, but real, and it surprises people: the rate limiter **fails closed**. If its counter errors, you get `503` with code `rate_limiter_unavailable` and `Retry-After: 5` rather than being let through. Treat it exactly like a `429`: back off and retry.

401 — and the brute-force throttle behind it

Bad keys get `401 unauthorized`. Note that **repeated auth failures from one IP are throttled**: past 30 failed attempts in a minute you get `429` instead of `401`. So a retry loop wrapped around a wrong key digs itself deeper. Fail fast on `401` — it will never fix itself by retrying.

422 — validation

`422 validation_failed` includes a `fields` array naming what was wrong:

```
{ "code": "validation_failed", "message": "Invalid email address.", "fields": ["email"] }
```

Surface `fields` to whoever is feeding you data. Retrying an unchanged `422` always fails again.

A retry policy that works

Status	Retry?	How
`429`, `503`	Yes	Honor `Retry-After`.
`500`	Yes	Exponential backoff, and use an `Idempotency-Key` on creates.
`401`, `403`, `404`, `405`, `413`, `422`	**No**	Fix the request or the key.
`409 idempotency_conflict`	Yes, shortly	Your own identical create is still in flight.

Full code table: `REFERENCE.md` — §6 Errors.

Step 5 — Page through lists

`page` starts at 1; `per_page` defaults to 50 and is **capped at 100** (higher values are silently clamped, not rejected). Loop until you reach `total_pages`:

```
page = 1
while True:
    data, meta = call("GET", "/customers", params={"page": page, "per_page": 100})
    for row in data:
        handle(row)
    pg = meta["pagination"]
    if pg["page"] >= pg["total_pages"]:
        break
    page += 1
```

Two practical notes. A large account is thousands of customers, so a full sync is dozens of calls — budget it against your rate limit rather than firing them all at once. And filter server-side where you can (`customer_id`, `status`, and for the schedule `from`/`to`/`type`) instead of pulling everything and filtering in your own code.

Step 6 — Write data safely

Creates and updates exist for customers, properties, and items & services.

Always send an Idempotency-Key on creates

A network timeout on a `POST` leaves you genuinely unsure whether the record was created. Send a UUID you generate:

```
curl -s -X POST https://secure.lawnprosoftware.com/api/v1/customers \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: 7c2b9f04-2f1a-4e3a-9a1d-6f5e2b8c1d22" \
  -d '{
        "first_name": "Jane",
        "last_name": "Doe",
        "email": "jane@example.com",
        "address": "123 Oak St",
        "city": "Birmingham",
        "state": "AL",
        "postal_code": "35203"
      }'
```

Retry with the **same** key and you get the original response replayed — `Idempotent-Replayed: true`, no duplicate row. Use a **fresh** key for a genuinely new record. Keys are scoped per endpoint, and failed creates release the key so you can cleanly retry rather than replaying an error forever.

Updates are partial

`PUT` (and `PATCH`) change **only the fields you send**. Omitted fields are left alone, so you never need to read-modify-write a whole object:

```
curl -s -X PUT https://secure.lawnprosoftware.com/api/v1/customers/42 \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -d '{ "phone": "205-555-0199" }'
```

Unknown fields in a body are ignored, not rejected — which means a typo'd field name fails **silently**. If a write seems to do nothing, check your spelling against `REFERENCE.md` — §12 Resources.

Deletes are soft

`DELETE` returns `{ "id": 42, "deleted": true }` and soft-deletes the record. It disappears from the API; it is not destroyed in LawnPro.

Bulk imports: suppress the automations

LawnPro-specific, and the mistake is expensive. Creating a customer fires that company's "customer added" automations — which can mean a real welcome email or text to a real person. Importing 500 customers with the defaults sends 500 messages.

Pass `"trigger_automations": false` on every create in an import:

```
{ "first_name": "Jane", "last_name": "Doe", "trigger_automations": false }
```

It must be a **real JSON boolean**. The check is strict (`=== false`), so `"false"` as a string, `0`, or `null` all count as "fire the automations." Verify with one test customer before running the batch.

Other validation worth knowing up front

- A customer needs **at least one** of `company_name`, `first_name`, `last_name`.
- A property's `customer_id` must be a customer in **your** account, or you get `422`.
- An item's `category_id` must be yours; if you omit it on create, it defaults to the account's first category, and the create **fails** if no category exists yet.
- Creating customers is subject to the account's plan limit — over it, you get `403 forbidden` rather than a validation error.

Step 7 — Receive webhooks instead of polling

Polling for "did anything change" wastes your rate limit and is always a little stale. Webhooks push the event to you when it happens. Register one HTTPS endpoint per system that needs events.

Full spec: `WEBHOOKS.md`. The short version:

Register the endpoint

In the app: **Settings → Webhooks → Add endpoint** (owner only). Or via API with `webhooks:write`:

```
curl -s -X POST https://secure.lawnprosoftware.com/api/v1/webhooks \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -d '{
        "url": "https://example.com/hooks/lawnpro",
        "events": ["invoice_paid_in_full", "customer_added"],
        "description": "Sync to our CRM"
      }'
```

The response contains `data.secret` (`whsec_…`) **once**. Store it now — it's never returned again, and rotating means deleting the endpoint and creating a new one.

> **Two different secrets.** The **API key** (`lp_live_…`) authenticates your calls *to* LawnPro. The **signing secret** (`whsec_…`) verifies deliveries *from* LawnPro. They are not interchangeable. If you only use the UI, you don't need an API key for webhooks at all.

Subscribe to specific event names, or `"*"` for everything. The authoritative list is the event picker in **Settings → Webhooks**; the catalog is in `WEBHOOKS.md` — §7.

What arrives

`POST`, JSON, and you must answer within **10 seconds**:

```
{
  "id": "evt_9f8c1b2a3d4e5f60718293a4b5c6d7e8",
  "event": "invoice_paid_in_full",
  "created_at": "2026-07-11T18:03:22Z",
  "company_id": 1234,
  "object": "invoice",
  "object_id": 987,
  "data": { "id": 987, "customer_id": 42, "total": 250.00, "balance_due": 0.00 }
}
```

`data` is the affected object in the same shape the REST API returns it. If the record can't be fully loaded at send time it degrades to a minimal `{ "id": …, "customer_id": … }` — always enough to fetch the rest from the API, so your handler should tolerate the thin version.

The four rules for your receiver

Get these right and webhooks are boring, which is what you want.

1. Verify the signature over the raw body. The HMAC covers `"<timestamp>.<raw body>"`. If you parse the JSON and re-serialize it before hashing, whitespace differences will break the comparison and you'll chase it for an afternoon.

```
$payload   = file_get_contents('php://input');          // raw bytes, unparsed
$timestamp = $_SERVER['HTTP_X_LAWNPRO_TIMESTAMP'] ?? '';
$sigHeader = $_SERVER['HTTP_X_LAWNPRO_SIGNATURE'] ?? '';
$expected  = 'sha256=' . hash_hmac('sha256', $timestamp . '.' . $payload, $YOUR_SECRET);

if (!hash_equals($expected, $sigHeader) || abs(time() - (int) $timestamp) > 300) {
    http_response_code(400);
    exit('invalid signature');
}
```

Compare in **constant time** (`hash_equals`, `crypto.timingSafeEqual`, `hmac.compare_digest`) and reject timestamps more than 300 seconds old. Node and Python versions: `WEBHOOKS.md` — §4.

2. Acknowledge fast, work later. Return any `2xx` within 10 seconds. Queue the actual processing. If you do the CRM sync inline and it takes 12 seconds, LawnPro treats it as failed and retries — and now you're processing twice.

3. De-duplicate on `X-LawnPro-Delivery`. Delivery is **at least once**. The same event can arrive twice (your `2xx` got lost, a worker died after sending). That header is stable across retries of the same delivery, so record it and skip what you've seen.

4. Don't rely on order, and ignore unknown events. Deliveries are best-effort in order but **not guaranteed**, so key decisions off the object's own state or `created_at`, not arrival sequence. And tolerate event names you didn't subscribe to — the `ping` test event is delivered regardless of your subscription list.

When you fail

Non-2xx, timeout, connection error, or DNS failure all count as failure. LawnPro retries with backoff — 1 minute, 5 minutes, 30 minutes, 2 hours, 6 hours — for **6 attempts total** (the first plus five retries), about 8½ hours end to end. After the sixth the delivery is dead-lettered and never retried.

Watch `consecutive_failures` on the endpoint, or `GET /api/v1/webhooks/{id}/deliveries` for the last 20 attempts with the real HTTP status and error. An endpoint quietly failing for nine hours has already lost data.

Step 8 — Test your webhook receiver

Send a real test event. In **Settings → Webhooks**, the paper-plane button queues a `ping` through the actual pipeline — same signing secret, same SSRF checks, same delivery log. A `ping` that shows **delivered** is genuine proof, not a simulation. Owner only, active endpoints only, one per endpoint per minute, and **one attempt with no retries**, so a failed test stays failed. Allow up to a minute for the worker to pick it up.

Because it's really signed, `ping` is the cheapest way to exercise your signature-verification code before live data flows.

You cannot point an endpoint at `localhost`. The worker re-resolves the hostname at delivery time and refuses anything resolving to a private, loopback, or reserved address — so tunnel your dev machine (ngrok or similar) and register the HTTPS tunnel URL. While you're still sketching, a request bin like webhook.site

shows you full headers and body, and Svix Play will check an HMAC for you.

Go-live checklist

Your API client

- ☐ Key lives in a secrets manager or env var — never in source control, never client-side.
- ☐ Honors `Retry-After` on `429` **and** `503`.
- ☐ Does not retry `401`/`403`/`422` (and knows repeated `401`s get throttled).
- ☐ Sends `Idempotency-Key` on every create.
- ☐ Pages to `total_pages` rather than assuming one page.
- ☐ Logs `meta.request_id` on failures.
- ☐ Branches on `errors[0].code`, not on message text.
- ☐ Bulk imports send `"trigger_automations": false`.

Your webhook receiver

- ☐ Public HTTPS.
- ☐ Verifies the signature over the **raw** body, constant-time, with the 300-second window.
- ☐ Returns `2xx` in under 10 seconds; heavy work is queued.
- ☐ Idempotent, keyed on `X-LawnPro-Delivery`.
- ☐ Survives out-of-order and duplicate deliveries.
- ☐ Ignores unknown event names, including `ping`.
- ☐ `whsec_` secret stored at creation.
- ☐ Someone is alerted on `consecutive_failures` climbing.

Operationally

- ☐ A `ping` has been delivered successfully to the production URL.
- ☐ You know who to call: **205-369-7052**, 7am–7pm CST, 7 days.
- ☐ If you're migrating off the legacy `/api`, that surface is deprecated — see `REFERENCE.md` — §16.

Troubleshooting

What you see	Why	Fix
`401 unauthorized` on every call	Missing `Bearer ` prefix, truncated secret, or key was revoked	Re-check the header format; confirm the key shows **Active** in Settings → API Access
`429` immediately, with no traffic	Failed-auth throttle (30 bad auths/IP/minute)	Stop retrying, fix the key, wait for the window
`403 insufficient_scope`	Key wasn't granted that scope	New request with the right scopes — scopes aren't editable after issuance
`403 forbidden`	Account suspended, or over the plan's customer limit on a create	Check account standing / plan

What you see	Why	Fix
`403 https_required`	Called over plain HTTP	Use `https://`
`404` on a record you can see in the app	The record belongs to another company, or is soft-deleted	Confirm the ID came from this same key's account
`405 method_not_allowed`	Writing to a read-only resource (invoice, estimate, payment, schedule)	Not supported in v1
`422` naming `customer_id`	Referencing another company's customer	Use an ID from your own account
Write returns `200` but nothing changed	Misspelled field name — unknown fields are ignored silently	Check spelling against `REFERENCE.md` §12
Settings → API Access isn't there	Account isn't in the pilot	Contact support
Signature never verifies	Hashing re-serialized JSON instead of the raw body	Capture raw bytes before parsing
Webhook retried even though you returned 200	Took longer than 10s to answer	Acknowledge first, queue the work
Same event handled twice	Normal — delivery is at least once	De-duplicate on `X-LawnPro-Delivery`
Test event never arrives	Endpoint paused, URL not public, or under the 1/minute throttle	Check **Recent deliveries** for the real status code

When you contact support about a specific call, bring the `request_id`. It turns "the API is broken" into a one-minute lookup.

LawnPro-specific gotchas

Collected in one place, because these are what cost outside developers a day:

1. **Half the API is read-only.** Invoices, estimates, payments, and schedule cannot be written in v1.
2. **Creating a customer can send a real message.** Automations fire on create. Use `"trigger_automations": false` for imports.
3. **Statuses are integers, not strings.** `custom_status = 1` is active, `6` is a lead; invoice `2` is paid, `4` past due. Tables: `REFERENCE.md` — §14 Enumerations.
4. **`pay_method = 4` is not money.** Payment method 4 is "Customer Credit Balance," an internal credit application. Exclude it from revenue math.
5. **Access is a feature flag, not a plan tier.** API access is a pilot; webhooks have their own flag. Neither is something a merchant can switch on themselves.
6. **Only the account owner** can request keys or manage webhooks.
7. **The rate limiter fails closed** — expect `503` as well as `429`.
8. **Unknown request fields are ignored silently.** Typos don't error.
9. **Deletes are soft**, everywhere.
10. **The two customer-scoring events ignore `"*"`.** `customer_score_at_risk` and `customer_churn_risk_flagged` must be named explicitly (they're Plus-only and model-driven, not user actions), e.g. `["*", "customer_score_at_risk"]`.

LawnPro Public REST API v1 — Developer Reference

The complete reference for the LawnPro public REST API. For a step-by-step walkthrough of building an integration see `GUIDE.md`; for a fast overview see `README.md`; for the machine-readable contract see `openapi.yaml` (OpenAPI 3.1). This document is the authoritative, human-readable description of every endpoint, field, and behavior.

- **Base URL:** `https://secure.lawnprosoftware.com/api/v1`
- **Content type:** `application/json` (request and response)
- **Auth:** Bearer API key (see Authentication)
- **Versioning:** the version is in the path (`/api/v1`). Breaking changes ship under a new path (`/api/v2`); v1 stays stable.

Table of contents

1. Conventions

Topic	Rule
Field names	The public field names in this document are the contract. Internal database column names are **never** exposed and may differ.
IDs	All resource IDs are integers.
Dates/times	Timestamps are ISO-8601 **UTC** (`2026-06-29T13:45:00Z`). Date-only fields are `YYYY-MM-DD`. A zeroed/empty date is returned as `null`.
Money	JSON numbers rounded to 2 decimals (e.g. `125.00`). Never strings.
Booleans	Real JSON `true`/`false` in responses, even though some are stored internally as `1`/`2`.
Unknown fields	Unknown fields in a request body are ignored. Only documented fields are written.
Partial updates	`PUT`/`PATCH` are treated as partial updates: only the fields you send are changed. Omitted fields are left untouched.
Request size	Request bodies are capped at **1 MB**. Larger bodies get `413 payload_too_large`.

2. Authentication

Every request must send an API key as a Bearer token:

```
Authorization: Bearer lp_live_<prefix>.<secret>
```

A key has two parts joined by a dot:

- `<prefix>` — a public identifier (stored, indexed, safe to log).
- `<secret>` — the secret half. Only a bcrypt hash of it is stored, so a database leak cannot reveal usable keys.

How keys are issued

Keys are **requested by the merchant and approved by LawnPro staff**:

1. The **account owner** (`user_role = 1`) submits a request in **Settings → API Access → Request an API key**, giving it a label and describing what the integration does. Only one open request per company at a time; the owner can cancel a pending one.
2. **LawnPro staff approve** the request in the Control Center, setting its scopes and rate limit. The request shows as *Awaiting approval* until then, and staff may decline it.
3. The owner clicks **Generate key** on the approved row. The full key is displayed **once**, at that moment, and can never be retrieved again. If it's lost, request a new key and revoke the old one.

Key material is generated in the owner's own browser session at claim time, so **LawnPro staff never see the secret** — they approve the request, not the key. Only the public prefix and a bcrypt hash of the secret

are stored.

> **Availability:** the API Access page is gated by the `api_access` feature flag, not by plan tier — it is a pilot surface. If the page isn't visible, the account hasn't been enabled; contact support (see Support).

Auth failures

Situation	Response
No `Authorization` header / not a Bearer token	`401 unauthorized`
Malformed key (no `prefix.secret` shape)	`401 unauthorized`
Unknown prefix, wrong secret, or revoked key	`401 unauthorized`
More than 30 failed auth attempts from one IP in 60s	`429 rate_limited` + `Retry-After`
Company account suspended	`403 forbidden`
Plain HTTP in production	`403 https_required`

Failed authentication is throttled **per client IP** (30 per 60-second window) so a known prefix can't be brute-forced. A retry loop around a bad key will therefore start receiving `429` instead of `401` — never retry a `401`.

> Keys do not expire on a timer; they remain valid until revoked. Only keys in `active` status with no revocation timestamp authenticate.

---

3. Scopes & authorization

Each key is granted a set of **scopes** shaped as `resource:action`. A read scope never implies write.

Resource	Read scope	Write scope
Customers	`customers:read`	`customers:write`
Properties	`properties:read`	`properties:write`
Items & Services	`items:read`	`items:write`
Invoices	`invoices:read`	*(read-only — no write)*
Estimates	`estimates:read`	*(read-only — no write)*
Payments	`payments:read`	*(read-only — no write)*
Schedule	`schedule:read`	*(read-only — no write)*
Webhooks	`webhooks:read`	`webhooks:write`

> Invoices, estimates, payments, and schedule are **read-only** in v1. The `:write` scopes for those resources are reserved for future use and currently grant nothing.

A call that lacks the required scope returns `403 insufficient_scope` with a message naming the missing scope, e.g.:

```
{ "data": null, "meta": { "request_id": "req_..." },
  "errors": [ { "code": "insufficient_scope",
                "message": "This API key is missing the required scope: customers:write" } ] }
```

Least privilege: request only the scopes the integration actually needs. A read-only reporting tool should get only `*:read` scopes.

---

4. Tenant isolation

A key is permanently bound to exactly one company. Every query is automatically and irrevocably scoped to that company:

- You can only read or modify your own company's records.
- Cross-resource references are validated against your account. For example, creating a property with a `customer_id` that isn't yours returns `422 validation_failed`; fetching another company's record returns `404 not_found` (it simply doesn't exist as far as your key is concerned).

There is no way to widen a key's scope to another company.

---

5. The response envelope

Every response — success or error — uses the same envelope:

```
{
  "data":   <object | array | null>,
  "meta":   { "request_id": "req_..." },
  "errors": []
}
```

- `data` — the resource object, an array of objects (lists), or `null` on error.
- `meta.request_id` — a unique id for this call, also returned as the `X-Request-Id` response header. Include it when contacting support.
- `meta.pagination` — present on list responses (see Pagination).
- `errors` — empty `[]` on success; one or more error objects on failure.

Success (single object)

```
{
  "data": { "id": 42, "first_name": "Jane", "last_name": "Doe" },
  "meta": { "request_id": "req_2f1c9a7b" },
  "errors": []
}
```

Success (collection)

```
{
  "data": [ { "id": 42 }, { "id": 41 } ],
  "meta": {
    "request_id": "req_2f1c9a7b",
    "pagination": { "page": 1, "per_page": 50, "total": 213, "total_pages": 5 }
  },
  "errors": []
}
```

---

6. Errors

On failure, `data` is `null` and `errors` contains one or more objects:

```
{
  "data": null,
  "meta": { "request_id": "req_2f1c9a7b" },
  "errors": [
    { "code": "validation_failed", "message": "Invalid email address.", "fields": ["email"] }
  ]
}
```

Field	Description
`code`	Stable machine-readable error code (table below). Branch on this, not on `message`.
`message`	Human-readable explanation. May change wording over time.
`fields`	*(optional)* The request fields that failed validation.

Error codes

code	HTTP	Meaning
`invalid_json`	400	Body was not valid JSON.
`unauthorized`	401	Missing, malformed, invalid, or revoked API key.
`forbidden`	403	Account suspended.
`https_required`	403	Request was plain HTTP in production.
`insufficient_scope`	403	Key lacks the scope required for this call.
`not_found`	404	Resource doesn't exist in your account.

code	HTTP	Meaning
`method_not_allowed`	405	HTTP method not supported for this resource (e.g. creating a read-only resource).
`idempotency_conflict`	409	A create with this `Idempotency-Key` is still in flight. Retry shortly.
`payload_too_large`	413	Request body exceeded 1 MB.
`validation_failed`	422	Input failed validation (see `fields`).
`rate_limited`	429	Rate limit exceeded, or too many failed auth attempts from your IP. See `Retry-After`.
`server_error`	500	Unexpected server error. Safe to retry idempotently.
`rate_limiter_unavailable`	503	The rate limiter itself errored; the request was rejected (fail-closed). Honor `Retry-After` and retry.

---

7. HTTP status codes

Code	When
`200 OK`	Successful read, update, or delete.
`201 Created`	Successful create.
`400 Bad Request`	Invalid JSON body.
`401 Unauthorized`	Authentication failed.
`403 Forbidden`	Suspended account, missing scope, or HTTPS required.
`404 Not Found`	Resource not in your account.
`405 Method Not Allowed`	Unsupported method for the resource.
`409 Conflict`	Idempotency key still processing.
`413 Payload Too Large`	Body over 1 MB.
`422 Unprocessable Entity`	Validation error.
`429 Too Many Requests`	Rate limited, or auth-failure throttle tripped.
`500 Internal Server Error`	Server-side failure.
`503 Service Unavailable`	Rate limiter unavailable (fail-closed). Retry after `Retry-After`.

---

8. Rate limiting

Limits are enforced **per key, per minute** (fixed window). The default is **120 requests/minute**; support can raise it per key (up to 6000/min).

Every response includes:

```
X-RateLimit-Limit: 120
X-RateLimit-Remaining: 118
```

When you exceed the limit you get `429 rate_limited` plus:

```
Retry-After: 37
```

Wait the indicated number of seconds, then retry. Build clients to honor `Retry-After` and back off rather than hammering the endpoint.

> **The limiter fails closed.** If its counter hits a transient error, the request is **rejected**, not allowed through — you get `503` with code `rate_limiter_unavailable` and `Retry-After: 5`. The limiter is a security control, so it is never silently disabled. Clients must treat `503 rate_limiter_unavailable` the same as `429`: honor `Retry-After` and retry.

---

9. Pagination

List endpoints accept:

Query param	Default	Max	Notes
`page`	`1`	—	1-indexed.
`per_page`	`50`	`100`	Values above 100 are clamped to 100.

The `meta.pagination` block tells you how to page through results:

```
"pagination": { "page": 2, "per_page": 50, "total": 213, "total_pages": 5 }
```

Iterate until `page` reaches `total_pages`.

---

10. Idempotency (safe create retries)

Network failures make it unsafe to blindly retry a `POST` — you might create the same record twice. To retry safely, send an **`Idempotency-Key`** header on create calls with a unique value you generate (a UUID is ideal):

```
Idempotency-Key: 7c2b9f04-2f1a-4e3a-9a1d-6f5e2b8c1d22
```

Behavior:

- **First request** runs normally and the result is stored against the key.
- **A retry with the same key** replays the original stored response instead of creating a duplicate.

Replays carry the header `Idempotent-Replayed: true`.

- **Scope:** the key is scoped to the **endpoint** (method + path). The same key value used on a *different* endpoint is treated as a separate operation, not a replay.
- **Concurrency:** if two requests with the same key arrive at once, only one performs the create. The other receives `409 idempotency_conflict` while the first is still in flight (retry shortly), and a normal replay once it finishes.
- **Failures are not cached.** If a create returns an error (4xx/5xx), the key is released so you can retry the operation cleanly — you won't be stuck replaying a failure. Only successful (2xx) responses are stored for replay.

> Keep the same `Idempotency-Key` for all retries of one logical create; generate a fresh key for a genuinely new create.

---

11. Security & transport

- **HTTPS only.** Production rejects plain HTTP with `403 https_required` and sends `Strict-Transport-Security`. Always call `https://`.
- **Keep keys secret.** Treat the key like a password. Store it in a secrets manager / environment variable, never in client-side code or a public repo.
- **Rotate on exposure.** If a key may have leaked, have support revoke it and issue a new one. Revocation is immediate.
- **Scope minimally.** Use read-only keys wherever possible.
- **Audit trail.** Every API call is logged server-side with its `request_id`, so support can trace a specific request you reference.

---

12. Resources

Common conventions for all resources:

- List: `GET /{resource}` — paginated, returns a collection envelope.
- Get one: `GET /{resource}/{id}` — `404 not_found` if not yours.
- Create: `POST /{resource}` — `201` on success; supports `Idempotency-Key`.
- Update: `PUT /{resource}/{id}` — partial update; only sent fields change.
- Delete: `DELETE /{resource}/{id}` — soft delete; returns `{ id, deleted: true }`.

Read-only resources (invoices, estimates, payments, schedule) support only the List and Get operations.

Customers

Scopes: `customers:read`, `customers:write` · **Operations:** list, get, create, update, delete

Fields (response)

Field	Type	Notes
`id`	integer	
`company_name`	string	
`first_name`	string	
`last_name`	string	
`email`	string	
`phone`	string	
`mobile`	string	
`address`	string	
`address2`	string	
`city`	string	
`state`	string	
`postal_code`	string	
`county`	string	
`country`	string	ISO country code (defaults to `US`).
`customer_number`	string	Human-facing customer number.
`type_id`	integer|null	Customer type.
`status`	integer|null	See Customer status.
`tax_exempt`	boolean	
`created_at`	datetime|null	
`updated_at`	datetime|null	

Writable fields (create/update)

`company_name`, `first_name`, `last_name`, `email`, `phone`, `mobile`, `address`, `address2`, `city`, `state`, `postal_code`, `county`, `country`, `type_id`, `status`, `tax_exempt`, plus:

- `trigger_automations` *(boolean, default `true`)* — set `false` on bulk imports to suppress the "customer added" automations for that record.

Validation:

- Provide **at least one** of `company_name`, `first_name`, or `last_name`.
- `email`, if provided, must be a valid address.
- Subject to your plan's customer limit; exceeding it returns `403 forbidden`.

List filters

Query param	Notes
`status`	Filter by customer status (e.g. `1` active, `6` lead). `0` is honored.
`email`	Exact email match.

Examples

Create:

```
curl -s -X POST https://secure.lawnprosoftware.com/api/v1/customers \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: 8f14e45f-ceae-4d3b-9a1e-1a2b3c4d5e6f" \
  -d '{
        "first_name": "Jane",
        "last_name": "Doe",
        "email": "jane@example.com",
        "phone": "205-555-0100",
        "address": "123 Oak St",
        "city": "Birmingham",
        "state": "AL",
        "postal_code": "35203"
      }'
```

Update (partial):

```
curl -s -X PUT https://secure.lawnprosoftware.com/api/v1/customers/42 \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -d '{ "phone": "205-555-0199" }'
```

Properties

Scopes: `properties:read`, `properties:write` · **Operations:** list, get, create, update, delete

A property is a service location belonging to a customer.

Fields (response)

Field	Type	Notes
`id`	integer	
`customer_id`	integer	Owning customer.
`name`	string	Optional label.
`address`	string	
`address2`	string	
`city`	string	
`state`	string	

Field	Type	Notes
`postal_code`	string	
`county`	string	
`country`	string	
`latitude`	string|null	
`longitude`	string|null	
`size`	number|null	Lot/turf size.
`size_type`	integer|null	Unit of the size value.
`notes`	string	
`status`	integer|null	
`created_at`	datetime|null	
`updated_at`	datetime|null	

Writable fields (create/update)

`customer_id` *(required on create)*, `name`, `address`, `address2`, `city`, `state`, `postal_code`, `county`, `country` (default `US`), `size`, `size_type`, `notes`, `status`, `latitude`, `longitude`.

Validation: `customer_id` must reference a customer in your account.

List filters

Query param	Notes
`customer_id`	List only properties for a given customer.

Example

```
curl -s -X POST https://secure.lawnprosoftware.com/api/v1/properties \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -d '{ "customer_id": 42, "address": "123 Oak St", "city": "Birmingham",
        "state": "AL", "postal_code": "35203", "size": 7500 }'
```

Items & Services

Scopes: `items:read`, `items:write` · **Operations:** list, get, create, update, delete

Your catalog of line items / services.

Fields (response)

Field	Type	Notes
`id`	integer	
`name`	string	

Field	Type	Notes
`category_id`	integer|null	Service category.
`price`	number	
`quantity`	number|null	
`type`	integer|null	
`unit`	string	
`description`	string	
`tax1`	number	
`tax2`	number	
`material_cost`	number	

Writable fields (create/update)

`name` *(required on create)*, `category_id`, `price`, `quantity`, `type`, `unit`, `description`, `tax1`, `tax2`, `material_cost`.

Validation:

- `name` is required on create.
- `category_id`, if provided, must belong to your account. On create, if omitted, it defaults to your account's first category (a create fails with `422 validation_failed` if no category exists yet).
- On update you may change `category_id` alone.

List filters

Query param	Notes
`category_id`	List only items in a given category.

Invoices (read-only)

Scope: `invoices:read` · **Operations:** list, get

Fields (response)

Field	Type	Notes
`id`	integer	
`number`	integer|null	Invoice number.
`customer_id`	integer	
`property_id`	integer|null	
`po_number`	string	
`title`	string	
`date`	date|null	
`status`	integer|null	See Invoice status.

Field	Type	Notes
`subtotal`	number	
`discount`	number	
`tax`	number	
`total`	number	
`paid_amount`	number	
`balance_due`	number	`total − paid_amount`.
`notes`	string	
`terms`	string	
`updated_at`	datetime|null	

List filters

Query param	Notes
`customer_id`	Invoices for one customer.
`status`	Filter by invoice status (`0` honored).

Estimates (read-only)

Scope: `estimates:read` · **Operations:** list, get

Fields (response)

Field	Type	Notes
`id`	integer	
`number`	integer|null	
`customer_id`	integer	
`title`	string	
`date`	date|null	
`status`	integer|null	See Estimate status.
`subtotal`	number	
`discount`	number	
`tax`	number	
`total`	number	
`paid_amount`	number	Deposit/amount applied.
`invoice_id`	integer|null	Set once converted to an invoice.
`notes`	string	
`terms`	string	
`created_at`	datetime|null	

Field	Type	Notes
`updated_at`	datetime|null	

List filters

Query param	Notes
`customer_id`	Estimates for one customer.
`status`	Filter by estimate status (`0` honored).

Payments (read-only)

Scope: `payments:read` · **Operations:** list, get

Fields (response)

Field	Type	Notes
`id`	integer	
`customer_id`	integer|null	
`invoice_id`	integer|null	
`method_id`	integer|null	See Payment method.
`date`	date|null	
`amount`	number	
`applied_to_invoice`	number	
`credit_amount`	number	
`tip_amount`	number	
`is_online`	boolean	
`is_refunded`	boolean	
`notes`	string	

List filters

Query param	Notes
`customer_id`	Payments for one customer.
`invoice_id`	Payments applied to one invoice.

Schedule / Visits (read-only)

Scope: `schedule:read` · **Operations:** list, get

Returns visits/events. Cancelled visits are not exposed and cannot be fetched by id.

Fields (response)

Field	Type	Notes
`id`	integer	
`customer_id`	integer|null	
`property_id`	integer|null	
`type`	integer|null	See Event type.
`title`	string	
`description`	string	
`date`	date|null	Visit date.
`end_date`	date|null	For multi-day events.
`has_time`	boolean	Whether a specific time is set.
`time_start`	string|null	
`time_end`	string|null	
`is_completed`	boolean	
`is_recurring`	boolean	
`recurring_id`	integer|null	The recurring series, if any.
`invoice_id`	integer|null	Linked invoice, if any.
`cost`	number	

List filters

Query param	Notes
`customer_id`	Visits for one customer.
`type`	Event type (`1`=Visit, `2`=Task, `3`=Meeting, `4`=Maintenance).
`from`	Start date (`YYYY-MM-DD`), inclusive.
`to`	End date (`YYYY-MM-DD`), inclusive.

Webhooks (management)

Scopes: `webhooks:read`, `webhooks:write` · **Operations:** list, get, create, update, delete

Manage the HTTPS endpoints that receive real-time event notifications. For the delivery format your endpoint must handle, see Webhooks (receiving events).

Fields (response)

Field	Type	Notes
`id`	integer	
`url`	string	The HTTPS destination.

Field	Type	Notes
`events`	string[]	Subscribed event names, or `["*"]` for all events except the customer-scoring pair, which must be named outright.
`description`	string	Optional label.
`status`	string	`active` or `paused`. Paused endpoints receive nothing.
`consecutive_failures`	integer	Resets to 0 after any success.
`last_success_at`	datetime|null	
`last_failure_at`	datetime|null	
`created_at`	datetime|null	
`updated_at`	datetime|null	

On **create only**, the response also includes `secret` — the signing secret, shown **once**. Store it; it is never returned again.

Writable fields (create/update)

Field	Notes
`url`	*(required on create)* Public HTTPS URL. Private/loopback/reserved addresses are rejected with `422`.
`events`	*(required on create)* Array or CSV of event names, or `"*"` for all events. `"*"` does **not** cover `customer_score_at_risk` or `customer_churn_risk_flagged` — name those explicitly, optionally alongside `"*"` (e.g. `["*", "customer_score_at_risk"]`). Other names sent beside `"*"` are redundant and dropped. See the event catalog.
`description`	Optional, ≤ 255 chars.
`status`	*(update only)* `active` or `paused`.

Extra endpoint

- `GET /webhooks/{id}/deliveries` — the 20 most recent delivery attempts for one endpoint (a health view). Each item: `event`, `status`, `attempts`, `last_code`, `last_error`, timestamps.

Example — create

```
curl -s -X POST https://secure.lawnprosoftware.com/api/v1/webhooks \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -d '{
        "url": "https://example.com/hooks/lawnpro",
        "events": ["invoice_paid_in_full", "customer_added"],
        "description": "Sync to our CRM"
      }'
# → 201; data.secret is present ONCE — save it.
```

---

13. Webhooks (receiving events)

Webhooks push events to **your** server the moment they happen, so you don't have to poll the API. Register one or more HTTPS endpoints (via the management API or **Settings → Webhooks** in the app), choose the events you care about, and LawnPro will `POST` a signed JSON message to each endpoint when a matching event fires.

Delivery request

- Method: `POST` with `Content-Type: application/json`.
- Fired for every company that has a matching active endpoint — independent of whether the company uses Automations/Sequences.

Headers

Header	Description
`X-LawnPro-Event`	The event name (e.g. `invoice_paid_in_full`).
`X-LawnPro-Delivery`	Unique id for this delivery (stable across retries). Use it to de-duplicate.
`X-LawnPro-Timestamp`	Unix timestamp the signature was computed at.
`X-LawnPro-Signature`	`sha256=<hex>` — see Verifying.

Body (envelope)

```
{
  "id": "evt_9f8c1b2a3d4e5f60718293a4b5c6d7e8",
  "event": "invoice_paid_in_full",
  "created_at": "2026-07-11T18:03:22Z",
  "company_id": 1234,
  "object": "invoice",
  "object_id": 987,
  "data": { "id": 987, "customer_id": 42, "total": 250.00, "balance_due": 0.00, "...": "..." }
}
```

- `id` — unique event id, stable across retries of the same delivery.
- `event` — the event name (same as `X-LawnPro-Event`).
- `object` / `object_id` — the type and id of the affected record.
- `data` — the affected object in the **same shape the REST API returns** for it (a Customer, Invoice, Estimate, Payment, or Event). If the object can't be fully loaded at send time, `data` degrades to a minimal `{ "id": ..., "customer_id": ... }` — always enough to fetch full detail from the REST API.

Verifying the signature

Every request is signed with your endpoint's secret so you can prove it came from LawnPro and wasn't tampered with. The signature covers the **exact raw body**:

```
signature = HMAC_SHA256(secret, "<X-LawnPro-Timestamp>.<raw request body>")
```

Compare it (constant-time) against the hex in `X-LawnPro-Signature` (after the `sha256=` prefix). Reject the request if it doesn't match, or if the timestamp is older than a few minutes (guards against replay).

PHP

```
$payload   = file_get_contents('php://input');
$timestamp = $_SERVER['HTTP_X_LAWNPRO_TIMESTAMP'] ?? '';
$sigHeader = $_SERVER['HTTP_X_LAWNPRO_SIGNATURE'] ?? '';
$expected  = 'sha256=' . hash_hmac('sha256', $timestamp . '.' . $payload, $YOUR_SECRET);

if (!hash_equals($expected, $sigHeader) || abs(time() - (int) $timestamp) > 300) {
    http_response_code(400);
    exit('invalid signature');
}
http_response_code(200); // acknowledge
```

Node

```
const crypto = require("crypto");
function verify(rawBody, timestamp, sigHeader, secret) {
  const expected = "sha256=" + crypto.createHmac("sha256", secret)
    .update(timestamp + "." + rawBody).digest("hex");
  const ok = crypto.timingSafeEqual(Buffer.from(expected), Buffer.from(sigHeader));
  return ok && Math.abs(Date.now() / 1000 - Number(timestamp)) < 300;
}
```

Responding, retries & ordering

- **Respond fast with any `2xx`** (within 10 seconds) to acknowledge. The body is ignored.
- Any non-2xx, timeout, or network error is **retried with backoff**: 1 min, 5 min, 30 min, 2 h, 6 h — **6 attempts total** (the first plus five retries), spanning about 8½ hours. After the sixth attempt fails the delivery is **dead-lettered** (visible in the deliveries health view) and never retried.
- Deliveries may arrive **out of order** and, rarely, **more than once**. Treat handling as idempotent and de-duplicate on `X-LawnPro-Delivery` (or the body `id`).
- Deleting or pausing an endpoint stops (and retires) its pending deliveries.

Security

- Endpoints **must be HTTPS**. Private, loopback, and reserved addresses are rejected at save time and re-checked at delivery time (SSRF protection).
- The signing secret is shown once at creation. If it leaks, delete the endpoint and create a new one.

Event catalog

Event names mirror the automation triggers. Common ones:

Group	Events
Estimates	`estimate_create`, `estimate_sent`, `estimate_accept`, `estimate_declined`, `estimate_responded`, `estimate_not_responded`, `estimate_expiring`

Group	Events
Invoices	`invoice_created`, `invoice_sent`, `invoice_past_due`, `before_invoice_past_due`, `invoice_paid_in_full`, `invoice_partially_paid`
Payments	`payment_to_account`, `payment_with_tip`, `payment_on_portal_by_customer`, `payment_added_to_invoice`, `payment_receipt_sent`, `credit_card_failed`
Visits	`visit_added`, `visit_completed`, `visit_skipped`, `visit_cancelled`, `before_visit`, `visit_note_added`
Customers	`customer_added`, `customer_reactivated`, `customer_cancelled`, `customer_added_credit_card`, `credit_card_expiring`
Reviews	`customer_left_review`, `customer_left_good_review`, `customer_left_bad_review`
Work Requests	`work_request_received`, `work_request_converted`

Subscribe to `"*"` to receive **all** events. The authoritative, always-current list is the event picker in **Settings → Webhooks**.

---

14. Enumerations

Integer code fields map to the following values.

Customer status

Value	Meaning
0	Inactive
1	Active
2	Written Off (balance remaining)
5	Pending Approval
6	Lead

Invoice status

Value	Meaning
1	Draft
2	Paid
3	Partial
4	Past Due
5	Pending
6	Written Off

Estimate status

Value	Meaning
1	Draft

Value	Meaning
2	Accepted
3	Declined
4	Invoiced
5	Pending
6	Changes Requested
7	Changes Updated

Event type

Value	Meaning
1	Visit to Customer
2	Task
3	Meeting
4	Maintenance

Payment method

Value	Method	Value	Method
1	Cash	11	Venmo
2	Check	12	Zelle
3	Credit Card	13	CashApp
4	Customer Credit Balance	14	Apple Pay
5	ACH Bank Transfer	15	Electronic Funds Transfer
6	Paypal	16	Debit
7	Other	…	(additional internal methods exist)
8	Debit Card		
9	Direct Debit		
10	BACS		

---

15. Code samples

curl

```
# List active customers (page 1)
curl -s "https://secure.lawnprosoftware.com/api/v1/customers?status=1&per_page=50" \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>"
```

PHP

```php
<?php
$key = getenv('LAWNPRO_API_KEY'); // lp_live_<prefix>.<secret>
$ch  = curl_init('https://secure.lawnprosoftware.com/api/v1/customers');
curl_setopt_array($ch, [
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER     => [
        'Authorization: Bearer ' . $key,
        'Content-Type: application/json',
        'Idempotency-Key: ' . bin2hex(random_bytes(16)),
    ],
    CURLOPT_POST       => true,
    CURLOPT_POSTFIELDS => json_encode([
        'first_name' => 'Jane',
        'last_name'  => 'Doe',
        'email'      => 'jane@example.com',
    ]),
]);
$response = json_decode(curl_exec($ch), true);
curl_close($ch);

if (!empty($response['errors'])) {
    throw new RuntimeException($response['errors'][0]['message']);
}
echo "Created customer #" . $response['data']['id'];
```

Python (requests)

```
import os, uuid, requests

BASE = "https://secure.lawnprosoftware.com/api/v1"
key  = os.environ["LAWNPRO_API_KEY"]  # lp_live_<prefix>.<secret>
headers = {"Authorization": f"Bearer {key}"}

# Create a customer (idempotent)
resp = requests.post(
    f"{BASE}/customers",
    headers={**headers, "Idempotency-Key": str(uuid.uuid4())},
    json={"first_name": "Jane", "last_name": "Doe", "email": "jane@example.com"},
    timeout=30,
)
body = resp.json()
if body["errors"]:
    raise RuntimeError(body["errors"][0]["message"])
customer_id = body["data"]["id"]

# Page through all properties for that customer
page = 1
while True:
    r = requests.get(f"{BASE}/properties",
                     headers=headers,
                     params={"customer_id": customer_id, "page": page, "per_page": 100},
                     timeout=30).json()
    for prop in r["data"]:
        print(prop["id"], prop["address"])
    pg = r["meta"]["pagination"]
    if pg["page"] >= pg["total_pages"]:
        break
    page += 1
```

JavaScript (Node, fetch)

```
const BASE = "https://secure.lawnprosoftware.com/api/v1";
const key  = process.env.LAWNPRO_API_KEY; // lp_live_<prefix>.<secret>

async function createCustomer(data) {
  const res = await fetch(`${BASE}/customers`, {
    method: "POST",
    headers: {
      "Authorization": `Bearer ${key}`,
      "Content-Type": "application/json",
      "Idempotency-Key": crypto.randomUUID(),
    },
    body: JSON.stringify(data),
  });

  if (res.status === 429) {
    const wait = Number(res.headers.get("Retry-After") || 1);
    await new Promise(r => setTimeout(r, wait * 1000));
    return createCustomer(data); // retry after backoff
  }

  const body = await res.json();
  if (body.errors.length) throw new Error(body.errors[0].message);
  return body.data;
}
```

---

16. Legacy API & deprecation

The older `/api` surface (which transmitted the account email/password in request headers) is **deprecated**. It still functions for now, but:

- Responses to legacy password-auth calls carry `Deprecation: true` and `Link: </api/v1>; rel="successor-version"`.
- It transmits credentials on every call and lacks scopes, rate limiting, and idempotency.

All new integrations must use `/api/v1`. Migrate existing ones to key-based auth as soon as practical.

---

17. Support & changelog

- **Get a key / raise limits / report an issue:** contact LawnPro support.
  - Phone: **205-369-7052** (7am–7pm CST, 7 days a week)
- When reporting a problem with a specific call, include its `request_id` (from the response body or the `X-Request-Id` header).

Changelog

Version	Notes
`v1` (1.0.0)	Initial public release: customers, properties, items (read/write); invoices, estimates, payments, schedule (read-only). Bearer key auth, scopes, per-key rate limiting, idempotent creates.
`v1` (1.1.0)	Added **Webhooks**: manage subscriptions via `/api/v1/webhooks` (scopes `webhooks:read`/`webhooks:write`) and receive signed, retried event deliveries at your own HTTPS endpoints. Also configurable in **Settings → Webhooks**.

LawnPro Webhooks — Developer Reference

Outbound webhooks push LawnPro business events to your own HTTPS endpoint the moment they happen, so an integration never has to poll `/api/v1`. This is the complete, self-contained technical spec: how to register an endpoint, the delivery format, how to verify authenticity, and the retry/health model.

> This document is the consolidated, shareable version of the webhook material in `REFERENCE.md` (§12 management, §13 receiving) and the machine-readable `openapi.yaml`. If any of these disagree, `openapi.yaml` and the running code are the source of truth.
>
> For a step-by-step walkthrough that builds a receiver from scratch — including the four rules that keep webhooks boring — see `GUIDE.md`.

- App UI: **Settings → Webhooks** (owner-only)
- Management API base: `https://secure.lawnprosoftware.com/api/v1/webhooks`
- Availability: by request, any plan (its own `webhooks` gate — you do **not** need API Access enabled, though having it also grants webhooks)

> **Two different secrets — don't confuse them:**
>
> - The **API key** (`lp_live_<prefix>.<secret>`) authenticates *your* calls to the management API (the `Authorization: Bearer …` header in every curl example below). See **Getting an API key**.
> - The **signing secret** (`whsec_…`) is per-endpoint and is used to *verify* the webhook deliveries LawnPro sends you. See §4.
>
> If you only manage webhooks in the UI (**Settings → Webhooks**), you don't need an API key at all — just create the endpoint and copy its `whsec_` secret.

---

0. Getting an API key

The webhook **management API** is part of LawnPro's REST API, so it uses the same Bearer key as everything else under `/api/v1`:

```
Authorization: Bearer lp_live_<prefix>.<secret>
```

Keys are **requested by the account owner and approved by LawnPro staff**. The owner submits a request in **Settings → API Access** naming the integration, staff approve it with the scopes it needs — for webhooks that's `webhooks:read` and/or `webhooks:write` — and the owner then clicks **Generate key**. The full key is shown **once** and can't be retrieved later; if it's lost, request a new one and revoke the old. The secret is generated in the owner's own session, so staff never see it.

Full details, auth failures, and scope rules: see REFERENCE.md — §2 Authentication and REFERENCE.md — §3

Scopes & authorization. To reach support, see REFERENCE.md — §17 Support & changelog — phone **205-369-7052**, 7am–7pm CST.

---

1. Concepts

An **endpoint** is one HTTPS URL you register, plus the list of events it subscribes to and a per-endpoint **signing secret**. When a subscribed event fires for your company, LawnPro enqueues a **delivery** (one per matching endpoint) and a background worker `POST`s the signed JSON envelope to your URL, retrying on failure.

- Fan-out is **per company** and tenant-isolated — an endpoint only ever receives events from the company that owns it.
- Webhooks fire for the business event **regardless** of whether the company uses Automations/Sequences.
- Delivery is **at least once**, best-effort in-order but **not guaranteed** in order. Design consumers to be idempotent.

---

2. Registering an endpoint

Two equivalent surfaces:

- **UI:** Settings → Webhooks → **Add endpoint** (account owner only).
- **API:** `POST /api/v1/webhooks` with scope `webhooks:write`.

Either way, the **signing secret is returned exactly once** at creation (prefix `whsec_`). It is never retrievable again; store it immediately. To rotate, delete the endpoint and create a new one.

Create (API)

```
curl -s -X POST https://secure.lawnprosoftware.com/api/v1/webhooks \
  -H "Authorization: Bearer lp_live_<prefix>.<secret>" \
  -H "Content-Type: application/json" \
  -d '{
        "url": "https://example.com/hooks/lawnpro",
        "events": ["invoice_paid_in_full", "customer_added"],
        "description": "Sync to our CRM"
      }'
```

```
// 201 Created — data.secret is present ONCE.
{
  "data": {
    "id": 7,
    "url": "https://example.com/hooks/lawnpro",
    "events": ["invoice_paid_in_full", "customer_added"],
    "description": "Sync to our CRM",
    "status": "active",
    "consecutive_failures": 0,
    "last_success_at": null,
    "last_failure_at": null,
    "created_at": "2026-07-12T15:00:00Z",
    "updated_at": "2026-07-12T15:00:00Z",
    "secret": "whsec_...48 hex chars..."
  },
  "meta": { "request_id": "req_..." },
  "errors": []
}
```

Writable fields

Field	Rules
`url`	**Required on create.** Public **HTTPS** URL. Private/loopback/reserved addresses → `422` `validation_failed`.
`events`	**Required on create.** Array or CSV of event names, or `"*"` for all. Unknown names → `422`.
`description`	Optional, ≤ 255 chars.
`status`	Update only: `active` or `paused`. Paused endpoints receive nothing.

Management operations

Operation	Call	Scope
List	`GET /api/v1/webhooks`	`webhooks:read`
Get one	`GET /api/v1/webhooks/{id}`	`webhooks:read`
Recent deliveries	`GET /api/v1/webhooks/{id}/deliveries`	`webhooks:read`
Create	`POST /api/v1/webhooks`	`webhooks:write`
Update	`PUT /api/v1/webhooks/{id}`	`webhooks:write`
Delete (soft)	`DELETE /api/v1/webhooks/{id}`	`webhooks:write`

Endpoint object fields (all reads): `id`, `url`, `events[]`, `description`, `status`, `consecutive_failures`, `last_success_at`, `last_failure_at`, `created_at`, `updated_at`. `secret` appears **only** in the create response.

---

3. The delivery request

- Method: `POST`, `Content-Type: application/json`.

- Timeout: your endpoint must respond within **10 seconds**.

Headers

Header	Description
`X-LawnPro-Event`	Event name, e.g. `invoice_paid_in_full`.
`X-LawnPro-Delivery`	Unique delivery id, **stable across retries**. De-duplicate on this.
`X-LawnPro-Timestamp`	Unix timestamp the signature was computed at.
`X-LawnPro-Signature`	`sha256=<hex>` HMAC — see §4.
`User-Agent`	`LawnPro-Webhooks/1.0`

Body envelope

```
{
  "id": "evt_9f8c1b2a3d4e5f60718293a4b5c6d7e8",
  "event": "invoice_paid_in_full",
  "created_at": "2026-07-11T18:03:22Z",
  "company_id": 1234,
  "object": "invoice",
  "object_id": 987,
  "data": { "id": 987, "customer_id": 42, "total": 250.00, "balance_due": 0.00 }
}
```

Field	Meaning
`id`	Unique event id; identical across retries of the same delivery.
`event`	Event name (matches `X-LawnPro-Event`).
`created_at`	ISO-8601 UTC event time.
`company_id`	The company the event belongs to.
`object` / `object_id`	Type and id of the affected record.
`data`	The affected object in the **same shape the REST API returns** (Customer / Invoice / Estimate / Payment / Event). Degrades to a minimal `{ "id": ..., "customer_id": ... }` if the record can't be fully loaded at send time — always enough to fetch full detail from the API.

`object` values map to: `customer`, `invoice`, `estimate`, `payment`, `event`/`visit`. Unrecognized types yield the minimal `data` shape.

---

4. Verifying the signature

Every request is signed with your endpoint's secret over the **exact raw body**:

```
signature = HMAC_SHA256(secret, "<X-LawnPro-Timestamp>.<raw request body>")
```

Steps your receiver MUST perform:

1. Read the **raw** body bytes (do not re-serialize parsed JSON — whitespace differences will break the HMAC).
2. Compute `sha256=` + `hash_hmac('sha256', timestamp . '.' . rawBody, secret)`.
3. Compare to `X-LawnPro-Signature` in **constant time**.
4. Reject if it doesn't match, or if `abs(now - timestamp) > 300` seconds (replay guard).
5. Only then process, and return `2xx`.

PHP

```
$payload   = file_get_contents('php://input');
$timestamp = $_SERVER['HTTP_X_LAWNPRO_TIMESTAMP'] ?? '';
$sigHeader = $_SERVER['HTTP_X_LAWNPRO_SIGNATURE'] ?? '';
$expected  = 'sha256=' . hash_hmac('sha256', $timestamp . '.' . $payload, $YOUR_SECRET);

if (!hash_equals($expected, $sigHeader) || abs(time() - (int) $timestamp) > 300) {
    http_response_code(400);
    exit('invalid signature');
}

$event = json_decode($payload, true);
// ...handle $event idempotently, keyed on $_SERVER['HTTP_X_LAWNPRO_DELIVERY']...

http_response_code(200);
```

Node (Express, raw body)

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

// Capture the raw body for HMAC (express.json's `verify` hook):
app.use(express.json({ verify: (req, _res, buf) => { req.rawBody = buf; } }));

function verify(req, secret) {
  const ts  = req.get("X-LawnPro-Timestamp") || "";
  const sig = req.get("X-LawnPro-Signature") || "";
  const expected = "sha256=" + crypto
    .createHmac("sha256", secret)
    .update(ts + "." + req.rawBody.toString("utf8"))
    .digest("hex");
  let ok = false;
  try { ok = crypto.timingSafeEqual(Buffer.from(expected), Buffer.from(sig)); } catch (_) {}
  return ok && Math.abs(Date.now() / 1000 - Number(ts)) < 300;
}

app.post("/hooks/lawnpro", (req, res) => {
  if (!verify(req, process.env.LAWNPRO_WEBHOOK_SECRET)) return res.sendStatus(400);
  // ...handle req.body idempotently on req.get("X-LawnPro-Delivery")...
  res.sendStatus(200);
});
```

Python (Flask)

```
import hmac, hashlib, time
from flask import request, abort

def verify(secret: str) -> bool:
    ts  = request.headers.get("X-LawnPro-Timestamp", "")
    sig = request.headers.get("X-LawnPro-Signature", "")
    raw = request.get_data()  # bytes, unparsed
    expected = "sha256=" + hmac.new(
        secret.encode(), f"{ts}.".encode() + raw, hashlib.sha256
    ).hexdigest()
    if not hmac.compare_digest(expected, sig):
        return False
    try:
        return abs(time.time() - int(ts)) < 300
    except ValueError:
        return False
```

---

5. Responses, retries & delivery semantics

- **Acknowledge with any `2xx`** within 10s. The response body is ignored.
- **Failure** = non-2xx, timeout, connection error, or DNS failure. Failed deliveries are retried on an increasing backoff:

Attempt	Sent after the previous attempt failed
1	(immediately, when the event fires)
2	1 minute
3	5 minutes
4	30 minutes
5	2 hours
6	6 hours

- **Max 6 attempts** (so 5 retries, ~8½ hours of backoff end to end). After the 6th attempt fails the delivery is **dead-lettered** (terminal `dead` status, visible in the deliveries health view) and is not retried again.
- **Ordering is not guaranteed.** Handle events by their own `created_at` / object state, not arrival order.
- **At least once.** A delivery can arrive more than once (e.g. your 2xx was lost, or a worker crashed after sending). **De-duplicate on `X-LawnPro-Delivery`** (or the body `id`).
- **Pausing** an endpoint defers its pending deliveries; **deleting** it retires them. Deliveries to an inactive/deleted endpoint are dropped.

Delivery health

`GET /api/v1/webhooks/{id}/deliveries` (or the UI) returns the 20 most recent attempts:

Field	Meaning
`id`	Delivery row id.
`endpoint_id`	The endpoint.
`event`	Event name.
`delivery_id`	The `X-LawnPro-Delivery` value.
`status`	`pending` | `processing` | `delivered` | `dead`.
`attempts`	Attempts made so far.
`last_code`	Last HTTP status received (0 = no response).
`last_error`	Last error string, if any.
`created_at` / `updated_at`	Timestamps.

The endpoint object also carries `consecutive_failures` (resets to 0 on any success), `last_success_at`, and `last_failure_at` for at-a-glance health.

---

6. Security

- **HTTPS only.** Non-HTTPS URLs are rejected.
- **SSRF protection.** The destination is validated at save time *and* re-resolved and re-checked at delivery time; any host resolving to a private, loopback, or reserved address (IPv4 or IPv6, including embedded/mapped IPv4) is refused. Unresolvable hosts are refused, not risked. The resolved public IP is pinned for the request.
- **Per-endpoint secret**, shown once at creation (`whsec_` + 48 hex chars). Treat it like a password; rotate by recreating the endpoint.
- **Tenant isolation.** Endpoints and deliveries are scoped to the owning company on every read and write.
- **Management is owner-only** in the UI, and requires the `webhooks:write` scope via the API.

---

7. Event catalog

Subscribe by exact event name, or `"*"` for all events — with the one exception noted under the table. The authoritative, always-current list is the event picker in **Settings → Webhooks** (it mirrors the Automation trigger catalog).

Group	Events
Estimates	`estimate_create`, `estimate_draft`, `estimate_sent`, `estimate_service`, `estimate_accept`, `estimate_declined`, `estimate_changes`, `estimate_responded`, `estimate_not_responded`, `estimate_expiring`
Invoices	`invoice_created`, `invoice_sent`, `invoice_past_due`, `before_invoice_past_due`, `invoice_paid_in_full`, `invoice_partially_paid`, `invoice_service`
Payments	`payment_to_account`, `payment_with_tip`, `payment_on_portal_by_customer`, `payment_added_to_invoice`, `payment_receipt_sent`, `credit_card_failed`
Visits	`visit_added`, `visit_completed`, `visit_skipped`, `visit_cancelled`, `before_visit`, `visit_note_added`, `property_not_serviced`
Customers	`customer_added`, `customer_reactivated`, `customer_cancelled`, `customer_has_tag`, `customer_added_credit_card`, `customer_added_bank_account`, `customer_removed_credit_card`, `customer_removed_bank_account`, `customer_inactive_30_days`, `customer_inactive_60_days`, `customer_inactive_90_days`, `credit_card_expiring`
Reviews	`customer_left_review`, `customer_left_good_review`, `customer_left_bad_review`, `customer_no_review`
Work Requests	`work_request_received`, `work_request_converted`
Tasks & Meetings	`todo_created`, `todo_completed`, `todo_has_category`, `before_task`, `meeting_created`, `meeting_completed`, `meeting_has_category`, `before_meeting`
Timer	`timer_started`, `timer_stopped`
Tags	`tag_added`, `tag_removed`
Contracts & Recurring	`contract_expiring`, `recurring_job_created`, `recurring_job_cancelled`
Customer scoring *(Plus, explicit subscription only)*	`customer_score_at_risk`, `customer_churn_risk_flagged`

The two customer-scoring events are the exception to `"*"`. They are the only events that come from a nightly model rather than something a person did, so an endpoint must name them outright — a bare `"*"` will not receive them. This keeps a long-standing catch-all endpoint from suddenly firing whatever it feeds on a score change its owner never subscribed to. They are also emitted only for Plus accounts.

To get everything *and* the scoring events, send both — e.g. `["*", "customer_score_at_risk"]`. The scoring name is kept alongside the wildcard rather than collapsed into it. Any other name sent beside `"*"` is redundant and dropped, since `"*"` already covers it.

Some events are scan-based (e.g. "inactive N days", "expiring") and are evaluated on a schedule rather than in direct response to a user action. LawnPro suppresses duplicate deliveries for the same endpoint/event/object in those cases, but consumers should still de-duplicate on `X-LawnPro-Delivery`.

8. Testing your endpoint

Send a test event

In **Settings → Webhooks**, the paper-plane button on an endpoint row queues a one-off **`ping`** delivery. It goes through the real pipeline — same signing secret, same SSRF and timeout rules, same delivery log — so a `ping` that shows as **delivered** is genuine proof the endpoint works, not a simulation. Watch the result in **Recent deliveries** with the real HTTP status code.

Details worth knowing:

- Account **owner** only, and only on **active** (not paused) endpoints.
- Limited to **one test per endpoint per minute**.
- **One attempt, no retries** (unlike real events, which retry six times). A failed test stays failed — fix your receiver and send another.
- It arrives on the worker's next run, so allow up to about a minute.

The `ping` event

`ping` is **not** in the event catalog and no business logic ever emits it — it is only ever sent when someone presses the test button. It's delivered **regardless of your subscription list**, so your handler must tolerate an event name it didn't subscribe to (good practice anyway: ignore unknown events rather than erroring on them).

```
{
  "id": "evt_9f2c…",
  "event": "ping",
  "created_at": "2026-07-27T18:20:11Z",
  "company_id": 1234,
  "object": "",
  "object_id": 0,
  "data": {
    "test": true,
    "source": "settings",
    "message": "Test event from LawnPro. …"
  }
}
```

Because it carries a real signature, a `ping` is the easiest way to exercise your signature-verification code (§4) before any live data flows.

While you're still building

Point the endpoint at a hosted request bin — webhook.site is the quickest (it hands you a URL and shows the full headers and body), and Svix Play will also check an HMAC signature for you.

You **cannot** point an endpoint at `localhost` or any private/reserved address: the worker re-resolves the host at delivery time and refuses non-public destinations (§6). For local development, expose your machine through a tunnel such as ngrok and register that HTTPS URL.

9. Implementation checklist

- ☐ Endpoint is public **HTTPS**.
- ☐ Reads the **raw** body before parsing.
- ☐ Verifies `X-LawnPro-Signature` (constant-time) and the timestamp window.
- ☐ Returns `2xx` in **< 10s**; does heavy work **after** acknowledging (queue it).
- ☐ Is **idempotent**, keyed on `X-LawnPro-Delivery`.
- ☐ Tolerates **out-of-order** and **duplicate** deliveries.
- ☐ Stores the `whsec_` secret securely at creation (it's shown once).
- ☐ Monitors `consecutive_failures` / the deliveries view for health.
- ☐ Ignores unknown event names (including `ping`) instead of erroring.