AVISTRATA
AVIATION BUSINESS PLATFORM

Partner API

Complete integration guide · API version 1

How to connect your system to Avistrata: access and keys, reading and writing aircraft, maintenance, flight and inventory data, keeping in sync with change events and signed webhooks, limits, errors and worked examples in curl, Python and JavaScript.

At a glance	
Base address	https://login.mrotracker.com/api/v1
Format	JSON over HTTPS
Authentication	API key, sent as a Bearer token
Machine-readable description	https://login.mrotracker.com/api/v1/openapi.json (OpenAPI 3.1)
Support	support@avistrata.com · 703.520.5040
Document date	October 2026

Avistrata is a web-based maintenance, flight operations and safety system for Part 91, Part 135 and Part 145 operators, *built by Leesburg Aviation LLC in Leesburg, VA.*

© 2026 Leesburg Aviation LLC. All rights reserved.

Contents

1. Overview What the API is, how access works, what you can do

2. Getting started Onboarding steps and your first call

3. Authentication and security Keys, rotation, expiry, revocation, IP allowlist

4. Resources The standard resources and what they hold

5. Reading data Lists, paging, sorting, field selection, filters, one record

6. Writing data Create, safe retries with Idempotency-Key, update, delete

7. Keeping in sync Change events, the events feed, webhooks, signature checks, sync pattern

8. Limits Rates, page and batch sizes, retention

9. Errors Error format and every error code

10. Worked examples curl, Python and JavaScript

11. OpenAPI, testing and going live OpenAPI description, testing, go-live checklist

12. Versions, changes and support What can change and how to get help

1. Overview

The Avistrata Partner API lets an approved partner system read and, where agreed, change data in a customer's Avistrata organization: aircraft, squawks and discrepancies, maintenance due items, components, MEL deferrals, AOG events, work orders, task cards, flights and releases, parts, stock, customers and invoices.

How access works

- **Partners are set up by Avistrata.** There is no self-service sign-up. Leesburg Aviation agrees the scope with you and the customer, then configures your access.
- **One organization per access.** Each key belongs to one customer organization (and its sub-groups). If you integrate with several Avistrata customers, you get a separate access and key for each.
- **You act as a named user.** Every call runs as a specific user in the customer's organization, chosen by the customer and Avistrata. You can never see or change more than that person could in the app, and the app's own rules (roles, sign-off rules, required fields) apply to your writes exactly as they do on screen.
- **Resources and fields are agreed per access.** For each resource, your access may read, create, update or delete; and each resource has a fixed list of readable, writable and filterable fields. `GET /api/v1` and `/openapi.json` always tell you exactly what your key can use.
- **Everything is logged.** Each call is recorded (time, key, path, result, address) and every setup change is written to the customer's audit log.

What you can do

Capability	How
Discover your access	`GET /api/v1` and `GET /api/v1/openapi.json`
List and search records	`GET /api/v1/{resource}` with filters, sorting, paging, field selection
Read one record	`GET /api/v1/{resource}/{id}`
Create records	`POST /api/v1/{resource}` (one, or up to 100 at a time), safe to retry with Idempotency-Key
Update records	`PATCH /api/v1/{resource}/{id}`
Delete records	`DELETE /api/v1/{resource}/{id}` (only where agreed)
Stay in sync	`GET /api/v1/events` (polling), signed webhooks (push), or `updated_since` on supported resources

2. Getting started

Onboarding steps

1. **Agree the scope.** Tell us (`support@avistrata.com`) which customer organization, which data you nee whether you need to write anything. The customer confirms the request.
2. **We set up your access.** We create your partner record and the access for that organization: the user you act as, the resources and actions, your request rate, and optionally the IP addresses you call from and an end date.
3. **Receive your key.** We send the API key over a secure channel. It is shown to us only once and stored only as a one-way hash, so keep it safe; if it's lost we issue a new one.
4. **Optional: webhooks.** Give us an https address to receive change events. We send you a signing secret (also shown once).
5. **Check your access.** Call `GET /api/v1` and download `/openapi.json`. Build against what those return.
6. **Go live.** Start with a read-only sync, then turn on writes once the customer has checked the results.

Your first call

```
curl -s https://login.mrotracker.com/api/v1 \
  -H "Authorization: Bearer mrot_YOUR_KEY"
```

A successful answer looks like this (shortened):

```
{
  "data": {
    "api": "Avistrata API",
    "version": "v1",
    "partner": "Acme Sync",
    "organization": { "id": "6f1c…", "name": "Acme Aviation" },
    "rate_per_minute": 120,
    "key": { "prefix": "mrot_Ab12Cd3", "expires_at": null },
    "resources": [
      {
        "key": "aircraft",
        "label": "Aircraft",
        "permissions": { "read": true, "create": false, "update": true, "delete": false },
        "fields": ["id", "tail_no", "type_model", "aircraft_sn", "group_id", "airframe_hours",
"is_active"],
        "writable_fields": ["airframe_hours"],
        "filters": ["tail_no", "aircraft_sn", "is_active"],
        "updated_since": false,
        "events": true,
        "path": "/api/v1/aircraft"
      }
    ],
    "docs": "https://login.mrotracker.com/api/v1/openapi.json"
  }
}
```

3. Authentication and security

Sending your key

Every request must carry your key. Either header works; Bearer is preferred:

```
Authorization: Bearer mrot_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
X-API-Key: mrot_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Keys start with `mrot_`. A request without a valid key gets `401` and is not logged against your access.

Keeping keys safe

- Call the API only from your servers. **Never put a key in a browser, mobile app or anything a user can download.** The API answers browser pre-flight requests, but a key in front-end code is a key anyone can copy.
- Store keys in a secrets manager or encrypted configuration, never in source control or logs.
- Use HTTPS only (the API does not answer plain HTTP).
- Ask for separate keys for production and testing; each can be revoked on its own.

Rotation, expiry and revocation

- **Rotation:** ask us for a second key, switch your system over, then ask us to revoke the old one. Both work during the overlap.
- **Expiry:** a key can have an end date. After it, calls get `401 key_expired`.
- **Revocation:** a revoked key stops working within about a minute (`401 key_revoked`).
- **Suspension:** Avistrata or the customer can pause an access or a whole partner (`403 connection_suspended` / `partner_suspended`). Access can also have an end date (`403 connection_expired`).

IP allowlist

Your access can be limited to fixed addresses (single IPv4 addresses, IPv4 ranges like `198.51.100.0/24`, or IPv6 addresses). Calls from anywhere else get `403 ip_not_allowed`. Tell us your outbound addresses if you want this; we recommend it for production.

4. Resources

A resource is one kind of record, reached at `/api/v1/{resource}`. The list below is the standard set. Your access includes only what was agreed, and the exact fields can differ per customer, so always check `GET /api/v1`.

Resource	What it holds	Typical filters	updated_since
`aircraft`	Aircraft in the organization: tail number, type, serial, airframe hours, active	tail_no, aircraft_sn, is_active	—
`discrepancies`	Squawks and discrepancies, including whether the aircraft is grounded	aircraft_id, grounded	—
`maintenance_due`	Inspections and due items	aircraft_id	where available
`components`	Installed components: position, make, model, serial, install date	aircraft_id, serial_number, is_active	yes
`mel_deferrals`	MEL deferrals, open and closed	aircraft_id, status	—
`aog_events`	Aircraft-on-ground events	aircraft_id, status	—
`work_orders`	Work orders	aircraft_id, work_order_number	—
`task_cards`	Task cards on work orders	work_order_id, card_number, aircraft_tail_no	—
`flights`	Bookings / flights	aircraft_id, release_id	—
`flight_releases`	Dispatch releases	aircraft_id, status, release_number	where available
`parts`	Parts catalog	part_number, part_number_key, status	yes
`stock`	Stock on hand: part, location, serial / batch, condition	part_id, location_id, serial_number, condition	yes
`customers`	Billing customers	name, email	—
`invoices`	Invoices (the customer payment-link token is never returned)	customer_id, status, invoice_number, aircraft_id	yes

> **Records belong to the organization.** Each record has an organization field (`group_id`). You only ever see records of your organization and its sub-groups; anything else answers `404`, exactly as if it didn't exist.
>
> **Ids are strings.** Most ids are UUIDs. Treat them as opaque text.

Records you will never get

User accounts, sign-in data, audit history, confidential safety reports and Avistrata's own configuration are never available through the API, whatever the access.

5. Reading data

Lists

```
GET /api/v1/{resource}?limit=100&offset=0&order=tail_no.asc&fields=id,tail_no,airframe_hours
```

Parameter	Meaning	Default / limits
`limit`	How many records per page	100; at most 500
`offset`	How many to skip (paging)	0
`order`	`field.asc` or `field.desc`; any readable field. The record id is always added as a tie-breaker, so paging is stable	The resource's default sort, else id
`fields`	Comma-separated list of fields to return	All readable fields
`updated_since`	Only records changed at or after this ISO 8601 time (resources that support it)	—
any filter field	See Filters below	—

The answer has the records in `data` and paging details in `meta`:

```
{
  "data": [ { "id": "28b2…", "tail_no": "N123AB", "airframe_hours": 4211.6 } ],
  "meta": { "resource": "aircraft", "limit": 100, "offset": 0, "count": 1, "total": 1, "next_offset":
null }
}
```

Keep requesting with `offset = next_offset` until `next_offset` is `null`.

Filters

Only the resource's filter fields can be used (listed in `GET /api/v1` as `filters`). Write `field=value` for equals, or `field=op:value`:

Operator	Meaning	Example
(none) / `eq:`	equals	`tail_no=N123AB`
`neq:`	not equal	`status=neq:closed`
`gt:` `gte:` `lt:` `lte:`	greater / less than (numbers, dates)	`due_date=lte:2026-12-31`
`like:`	contains / pattern, not case-sensitive; `*` is the wildcard	`tail_no=like:N1*`
`in:`	one of a comma-separated list	`status=in:open,deferred`
`is:`	`null`, `true` or `false`	`grounded=is:true`

Several filters are combined with AND. A filter on a field that isn't allowed gets `400 bad_filter` (the message lists the allowed ones).

One record

```
GET /api/v1/aircraft/28b25bc1-a4ab-4c61-b2fa-7790df4e4068

{ "data": { "id": "28b2…", "tail_no": "N123AB", "type_model": "PC-12/47E", "airframe_hours": 4211.6, …
} }
```

Unknown id, or a record outside your organization: `404 not_found`.

6. Writing data

> Writes are only possible where your access allows the action **and** the field is on the resource's writable list. Everything else answers `400 bad_field` (with the list of writable fields) or `403 forbidden`.
>
> Writes run as the user you act as, so the app's own checks apply: a change that user couldn't make on screen is refused with `403`, and invalid values (for example a required field missing) with `422 invalid`.

Create

```
POST /api/v1/discrepancies
Content-Type: application/json
Idempotency-Key: sync-2026-10-08-N123AB-0042

{ "aircraft_id": "28b2…", "discrepancy": "Left landing light inop" }
```

- Send one object, or an array of up to **100** objects (created together).
- The organization field (`group_id`) is optional. Leave it out and the record goes to your organization; set it to one of its sub-groups if needed. Anything outside your organization is refused.
- Answer: `201` with the new record(s) in `data`, showing your readable fields (including the new id).

Retrying safely: Idempotency-Key

Networks fail. To make a create safe to retry, send a unique `Idempotency-Key` header (8 to 128 letters, digits, `- _ . :`). If we've already processed that key for your access in the last 24 hours, we return the original answer again (with header `Idempotent-Replay: true`) instead of creating a duplicate. Use a key that identifies the record in your system.

Update

```
PATCH /api/v1/aircraft/28b2…
Content-Type: application/json

{ "airframe_hours": 4213.1 }
```

Send only the fields you're changing. Answer: `200` with the updated record. Updates are not idempotency-keyed, but repeating the same PATCH gives the same result.

Delete

```
DELETE /api/v1/work_orders/7c1e…        →  204 No Content
```

Deletes are rarely granted. Prefer status changes (for example closing a record) where the resource offers them.

7. Keeping in sync

There are three ways to find out what changed. Most integrations use webhooks for speed and the events feed or `updated_since` as a safety net.

Method	How it works	Best for
Webhooks	We POST signed change events to your https address within about a minute	Near-real-time reactions
Events feed	`GET /api/v1/events?after=<last id>` returns changes in order	Polling, catching up after downtime
updated_since	`GET /api/v1/{resource}?updated_since=<time>` on resources that support it	Simple periodic syncs

Change events exist only for resources where Avistrata has switched them on for your access (`events: true` in `GET /api/v1`). Events are kept for 30 days.

Events

Each event says what changed, not the new values. Fetch the record with GET to get its current state (your field permissions always apply).

```
{
  "id": 18235,
  "type": "aircraft.updated",
  "resource": "aircraft",
  "record_id": "28b25bc1-a4ab-4c61-b2fa-7790df4e4068",
  "changed": ["airframe_hours"],
  "occurred_at": "2026-10-08T18:22:51.402Z"
}
```

- `type` is `{resource}.created`, `{resource}.updated` or `{resource}.deleted`.
- `changed` lists the columns that changed (updates only). It can include columns you can't read; ignore those.
- `id` increases over time. Store the highest id you've processed.

Polling the events feed

```
GET /api/v1/events?after=18200&limit=500&resource=aircraft,discrepancies

{ "data": [ …events… ], "meta": { "after": 18200, "next_after": 18235, "more": false } }
```

Call again with `after = next_after`. While `more` is true, call straight away; otherwise wait (every 1 to 5 minutes is plenty). `limit` is at most 500.

Webhooks

Each delivery is a POST with up to 100 events:

```
POST https://your-system.example.com/mro/webhook
Content-Type: application/json
User-Agent: Avistrata-Webhooks/1
X-MRO-Webhook-Id: 78cb1db8-2bf5-4648-8b14-7a7f33037057
X-MRO-Signature: t=1791483713,v1=b3fb0c…

{ "webhook_id": "78cb1db8-…", "events": [ { "id": 18235, "type": "aircraft.updated", … } ] }
```

- **Answer with any 2xx within 8 seconds.** Do the real work afterwards (queue it).
- **Failures are retried** with growing gaps: 1, 2, 4, 8, 16, 32 and then every 60 minutes, starting from the first undelivered event, so nothing is skipped. After 50 failures in a row the webhook is switched off and we contact you.
- **Deliveries can repeat** (for example if your 2xx was lost). De-duplicate by event `id`.
- Events arrive in `id` order within a webhook. You can choose all events or specific ones (for example `aircraft.updated` or `work_orders.*`).
- A `ping` event (id 0) is sent when we test your webhook.

Checking the signature

Always check `X-MRO-Signature` before trusting a delivery:

1. Split the header into `t` (unix seconds) and `v1` (hex).
2. Compute HMAC-SHA256 of the string `t + "." + raw request body` using your webhook secret.
3. Compare it with `v1` using a constant-time comparison.
4. Reject it if `t` is more than 5 minutes from your clock (replay protection).

Use the raw body bytes exactly as received, before any JSON parsing.

Node.js (Express)

```
const crypto = require("crypto");
app.post("/mro/webhook", express.raw({ type: "application/json" }), (req, res) => {
  const sig = req.get("X-MRO-Signature") || "";
  const parts = Object.fromEntries(sig.split(",").map(p => p.split("=")));
  const expected = crypto.createHmac("sha256", process.env.MRO_WEBHOOK_SECRET)
    .update(parts.t + "." + req.body).digest("hex");
  const ok = parts.v1 && parts.v1.length === expected.length &&
    crypto.timingSafeEqual(Buffer.from(parts.v1, "hex"), Buffer.from(expected, "hex")) &&
    Math.abs(Date.now() / 1000 - Number(parts.t)) < 300;
  if (!ok) return res.status(400).send("bad signature");
  const { events } = JSON.parse(req.body);
  queue.push(events);          // process after answering
  res.sendStatus(204);
});
```

Python (Flask)

```
import hmac, hashlib, time, json, os
from flask import Flask, request, abort
app = Flask(__name__)

@app.post("/mro/webhook")
def mro_webhook():
    raw = request.get_data()
    parts = dict(p.split("=", 1) for p in request.headers.get("X-MRO-Signature", "").split(",") if "="
in p)
    expected = hmac.new(os.environ["MRO_WEBHOOK_SECRET"].encode(),
                        (parts.get("t", "") + ".").encode() + raw, hashlib.sha256).hexdigest()
    if not hmac.compare_digest(expected, parts.get("v1", "")) or abs(time.time() - int(parts.get("t",
0"))) > 300:
        abort(400)
    events = json.loads(raw)["events"]
    enqueue(events)             # process after answering
    return "", 204
```

Recommended sync pattern

1. **First load:** page through each resource (`limit=500`, sorted by id) and store the records with their ids.
2. **Record your starting point first:** before the first load, read `GET /api/v1/events?after=0` until `more` is false and store `next_after`. Changes made during your first load then still reach you.
3. **Stay current:** on each webhook (or each poll of `/events`), fetch the records named in the events and update your copy; remove deleted ones.
4. **Catch up:** if you were offline, call `/events?after=<last processed id>` until `more` is false. Events are kept 30 days; after a longer gap, repeat the first load.
5. **Reconcile nightly** (optional): compare counts or use `updated_since` where supported.

© 2026 Leesburg Aviation LLC. All rights reserved.

8. Limits

Limit	Value	What happens
Requests per minute	Set per access, 120 by default	`429 rate_limited` with `Retry-After` (seconds until the next minute)
Records per page	500 (default 100)	Larger `limit` values are reduced to 500
Records per create	100	`400 bad_body`
Request body	1 MB	`413 too_large`
Webhook answer time	8 seconds	Counted as a failure and retried
Events kept	30 days	Older events are gone; resync with a full load
Idempotency keys kept	24 hours	After that, the same key creates a new record

Every answer has `X-RateLimit-Limit` (your per-minute limit) and `X-Request-Ms` (how long we took). On `429`, wait `Retry-After` seconds; spread bulk work out rather than bursting.

 © 2026 Leesburg Aviation LLC. All rights reserved. ·

9. Errors

Errors always have the same shape, with a stable `code` for your program and a `message` for people:

```
HTTP/1.1 400 Bad Request
{ "error": { "code": "bad_filter", "message": "Can't filter aircraft by \"group_id\". Filters:
tail_no, aircraft_sn, is_active." } }
```

HTTP	code	Meaning and what to do
400	`bad_json`	The body isn't valid JSON
400	`bad_body`	Wrong body shape (object or array of 1 to 100 for POST; object for PATCH)
400	`bad_field`	A field you can't read or write; the message lists the allowed ones
400	`bad_filter` / `bad_value` / `bad_order`	Filter field not allowed, bad value (e.g. date), or sort field not readable
400	`unsupported`	`updated_since` on a resource that doesn't support it
400	`bad_idempotency_key`	Idempotency-Key in the wrong format
401	`unauthorized` / `key_revoked` / `key_expired`	Missing, unknown, revoked or expired key
403	`forbidden`	Your access or the user you act as may not do this
403	`connection_suspended` / `connection_expired` / `partner_suspended`	Access paused or ended; contact us
403	`ip_not_allowed`	Call came from an address not on your allowlist
404	`not_found`	No such resource, record or path (or the record is outside your organization)
405	`method_not_allowed`	Wrong HTTP method for this path
413	`too_large`	Body over 1 MB
422	`invalid`	The app refused the values (a required field, a format, a business rule); see the message
429	`rate_limited`	Too many requests this minute; wait `Retry-After` seconds
502	`upstream_error`	The database couldn't complete the request; retry with back-off
503	`not_configured`	A resource isn't fully set up on our side; contact us
500	`server_error`	Unexpected; retry with back-off, then contact us with the time of the call

Retry rule of thumb: retry `429`, `502`, `500` and network errors with growing waits (for example 1, 2, 4, 8 seconds, max about a minute). Don't retry `4xx` other than `429` without changing the request. Use `Idempotency-Key` on creates so retries never duplicate.

10. Worked examples

curl

```
# Active aircraft, just id, tail and hours
curl -s "https://login.mrotracker.com/api/v1/aircraft?is_active=true&fields=id,tail_no,airframe_hours"
\
  -H "Authorization: Bearer $MRO_KEY"

# Grounding squawks for one aircraft
curl -s "https://login.mrotracker.com/api/v1/discrepancies?aircraft_id=28b2…&grounded=is:true" \
  -H "Authorization: Bearer $MRO_KEY"

# Parts changed since 1 October
curl -s "https://login.mrotracker.com/api/v1/parts?updated_since=2026-10-01T00:00:00Z&limit=500" \
  -H "Authorization: Bearer $MRO_KEY"

# Update airframe hours (if your access allows it)
curl -s -X PATCH "https://login.mrotracker.com/api/v1/aircraft/28b2…" \
  -H "Authorization: Bearer $MRO_KEY" -H "Content-Type: application/json" \
  -d '{"airframe_hours": 4213.1}'
```

Python: read everything, page by page

```
import os, time, requests
BASE = "https://login.mrotracker.com/api/v1"
S = requests.Session()
S.headers["Authorization"] = "Bearer " + os.environ["MRO_KEY"]

def get(path, **params):
    for attempt in range(6):
        r = S.get(BASE + path, params=params, timeout=30)
        if r.status_code == 429:
            time.sleep(int(r.headers.get("Retry-After", "5"))); continue
        if r.status_code >= 500:
            time.sleep(2 ** attempt); continue
        r.raise_for_status()
        return r.json()
    raise RuntimeError("Avistrata API kept failing")

def all_records(resource, **filters):
    offset = 0
    while offset is not None:
        page = get("/" + resource, limit=500, offset=offset, order="id.asc", **filters)
        yield from page["data"]
        offset = page["meta"]["next_offset"]

for ac in all_records("aircraft", is_active="true"):
    print(ac["tail_no"], ac.get("airframe_hours"))
```

JavaScript (Node 18+): follow the events feed

```
const BASE = "https://login.mrotracker.com/api/v1";
const H = { Authorization: "Bearer " + process.env.MRO_KEY };
let after = Number(await store.get("mro_last_event") || 0);

async function poll() {
  for (;;) {
    const r = await fetch(`${BASE}/events?after=${after}&limit=500`, { headers: H });
    if (r.status === 429) { await sleep(Number(r.headers.get("retry-after") || 5) * 1000); continue; }
    if (!r.ok) throw new Error((await r.json()).error.message);
    const { data, meta } = await r.json();
    for (const e of data) {
      if (e.type.endsWith(".deleted")) await local.remove(e.resource, e.record_id);
      else {
        const one = await fetch(`${BASE}/${e.resource}/${e.record_id}`, { headers: H });
        if (one.ok) await local.upsert(e.resource, (await one.json()).data);
      }
    }
    after = meta.next_after; await store.set("mro_last_event", after);
    if (!meta.more) break;
  }
}
setInterval(poll, 2 * 60 * 1000);
```

Creating with a safe retry

```
curl -s -X POST "https://login.mrotracker.com/api/v1/discrepancies" \
  -H "Authorization: Bearer $MRO_KEY" -H "Content-Type: application/json" \
  -H "Idempotency-Key: acme-squawk-99812" \
  -d '{"aircraft_id":"28b2…","discrepancy":"Pitot heat inop, found on preflight"}'
# Repeating exactly the same call returns the same record, with Idempotent-Replay: true
```

11. OpenAPI, testing and going live

OpenAPI description

`GET /api/v1/openapi.json` (with your key) returns an OpenAPI 3.1 description of exactly your access: the paths, methods, parameters, filters and fields you can use. Import it into Postman, Insomnia or a code generator. It changes when your access changes, so re-download it after we change your scope.

Testing

- Ask for a test access on a test organization (or a separate test key) before you touch live data.
- Use the `Send test` ping to check your webhook endpoint and signature check.
- We can see your calls and errors in our call log; send us the time and path of anything unexpected.

Go-live checklist

✓	Check
☐	Key stored in a secrets manager; not in code, logs or front-end
☐	All calls server-side over HTTPS; outbound IPs sent to us for the allowlist
☐	Paging until next_offset is null; sorted by id for full loads
☐	429 handled with Retry-After; 5xx retried with back-off
☐	Idempotency-Key on every create
☐	Webhook signature and timestamp checked; 2xx within 8 seconds; duplicates ignored by event id
☐	Last processed event id stored; catch-up via /events after downtime
☐	Unknown fields in answers ignored (we add fields over time)
☐	Customer has checked a sample of synced data before writes are turned on

© 2026 Leesburg Aviation LLC. All rights reserved.

12. Versions, changes and support

- This is version 1 (`/api/v1`). Within v1 we may **add** resources, fields, filters, event types and error codes. Build your integration to ignore fields and event types it doesn't know.
- Changes that would break existing integrations (removing or renaming fields, changing meanings) will only come in a new version, announced in advance to partners' contacts on file.
- A customer can change what you may access (for example remove a field or a resource). `GET /api/v1` always shows your current access.

Support

Email support@avistrata.com or call **703.520.5040**. Include your partner name, the key prefix (the first 12 characters, never the whole key), the time of the call (UTC), the method and path, and the error code. For security issues, write to security@avistrata.com .

> This guide describes the API as built in October 2026. The partner agreement between you, the customer and Leesburg Aviation LLC governs data use, confidentiality and service levels.