info@jana.earth

Jana Unified Environmental Data API -- User Guide

Version: 3.2.2 **Last Updated:** 2026-04-23 **Auth URL:** `https://auth-dev.jana.earth` (device-code login, token refresh) **Data URL:** `https://api-dev.jana.earth` (all data/ESG endpoints)

Table of Contents

Introduction

The Jana Unified Environmental Data API provides seamless access to air quality measurements, greenhouse gas emissions, corporate entity data, carbon budgets, severe weather events, and environmental analytics through a single interface. It integrates data from six authoritative sources:

Source	Data Type	Coverage
OpenAQ	Air quality measurements (30 parameters — PM2.5, PM10, PM1, O3, NO2, CO, SO2, temperature, relative humidity, particle counts, etc.)	50,514 monitoring stations across 159 countries; 311,602 sensors; ~1.49B measurements spanning 2016-01-30 → present
Climate TRACE	Asset-level CO2e emissions (per-gas CO2/CH4/N2O only	~153K named industrial facilities (power 9,732; manufacturing 33,685; waste 96,099; plus fossil-fuel ops, mining, F-gas sites) + gridded/polygon coverage of

Source	Data Type	Coverage
	via `/annual-country-emissions/`)	agriculture (1,002,010 field polygons), buildings (114,070 × 1 km² tiles), transportation (81,918 ships/roads/airports) = **1,337,615 "emitting assets"** in total; 53.4M emission records; 251 countries; 45+ sector slugs. Note: Climate TRACE's term "asset" covers any discrete emitter, not just plant-like facilities — see the Inventory section under "Climate TRACE -- Emissions Data" for the full two-category breakdown.
EDGAR	Emissions modeling, country-level totals, gridded data	Global coverage, 0.1 degree resolution grids
GLEIF	Legal Entity Identifiers (LEI), corporate ownership, reporting exceptions	3.25M+ legal entities, 637K+ relationships globally
GCP	National CO2 emissions, fossil fuel breakdown, carbon & methane budgets	60K+ national records, 273 fuel-type records (1750-2023)
NOAA	Severe weather events (tornadoes, hurricanes, floods, etc.)	US coverage, 2015-2024, spatial queries supported

API Architecture

The platform uses a **dual-URL** architecture:

URL	Purpose	Services
`https://auth-dev.jana.earth`	Authentication	Device-code flow, token refresh, user profile
`https://api-dev.jana.earth`	Data access	ESG unified, data source APIs, documentation

The API is organized into four sections:

- **ESG Unified API** (`/api/v1/esg/`) -- Cross-source queries, analytics, and export
- **Data Source APIs** (`/api/v1/data-sources/`) -- Source-specific endpoints with full detail
- **Authentication** (`/api/auth/`) -- Login, token management, user profile
- **Documentation** (`/api/docs/`, `/api/redoc/`) -- Interactive Swagger and ReDoc

---

Authentication

Jana uses **JWT (JSON Web Token)** authentication with two login methods. All authenticated requests use Bearer tokens in the `Authorization` header.

Device Code Flow (Recommended)

The device-code flow (RFC 8628) is the primary authentication method. It works like `aws sso login` -- you approve a login request in your browser, and the SDK receives tokens automatically. No passwords are stored or transmitted in your code.

How it works:

1. Your application requests a device code from the auth server
2. A browser window opens with a verification URL
3. You log in and click "Approve"
4. The SDK receives JWT access and refresh tokens

Using the Python SDK:

```
from eko_client import EkoUserClient

client = EkoUserClient(
    base_url="https://auth-dev.jana.earth",       # auth endpoints
    api_base_url="https://api-dev.jana.earth",     # data endpoints
```

```
    client_id="jana-sdk",
    timeout=120,
)

# Opens browser automatically -- log in and approve
client.login_device()

# Verify connection
health = client.get_health()
print(f"Connected: {health.get('status')}")
user = client.get_user_info()
print(f"Logged in as: {user.get('email')}")
```

Using curl (manual device-code flow):

```
# Step 1: Request device code
curl -X POST https://auth-dev.jana.earth/api/auth/device-code/ \
  -H "Content-Type: application/json" \
  -d '{"client_id": "jana-sdk"}'
```

Response:

```
{
  "device_code": "GmRhmhcxhwAzkoEqiMEg_DnyEysNkuNhszIySk9eS",
  "user_code": "WDJB-MJHT",
  "verification_uri": "https://auth-dev.jana.earth/activate",
  "expires_in": 900,
  "interval": 5
}
```

```
# Step 2: Open verification_uri in browser, enter user_code, log in

# Step 3: Poll for tokens (repeat every `interval` seconds)
curl -X POST https://auth-dev.jana.earth/api/auth/device-token/ \
  -H "Content-Type: application/json" \
  -d '{"device_code": "GmRhmhcxhwAzkoEqiMEg_DnyEysNkuNhszIySk9eS", "client_id": "jana-sdk"}'
```

Response (after approval):

```
{
  "access": "eyJhbGciOiJSUzI1NiIs...",
  "refresh": "eyJhbGciOiJSUzI1NiIs..."
}
```

```
# Step 4: Use the access token for all API calls
TOKEN="eyJhbGciOiJSUzI1NiIs..."
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/esg/summary/"
```

Password Login (Legacy/Scripting)

For scripts and automation where a browser isn't available, password login returns a DRF token.

```
curl -X POST https://auth-dev.jana.earth/api/auth/login/ \
  -H "Content-Type: application/json" \
  -d '{"username": "your-username", "password": "your-password"}'
```

Response:

```
{
  "token": "9a4f2c8d7b3e1a5f6c9d0e2b4a7f8c3d1e6b9a0c"
}
```

```
# Use with Token scheme (not Bearer)
curl -H "Authorization: Token 9a4f2c8d7b3e1a5f6c9d0e2b4a7f8c3d1e6b9a0c" \
  "https://api-dev.jana.earth/api/v1/esg/summary/"
```

Using the Python SDK:

```
client = EkoUserClient(
    base_url="https://auth-dev.jana.earth",
    api_base_url="https://api-dev.jana.earth",
)
client.login_password("your-email@example.com", "your-password")
```

Token Lifecycle

Token	Lifetime	Refresh
JWT access token	15 minutes	Auto-refreshed by SDK on 401/403
JWT refresh token	24 hours	Re-login required when expired
DRF token (password login)	No expiry	Valid until logout or revoked

The Python SDK handles token refresh automatically. If the refresh token itself expires, an `EkoSessionExpiredError` is raised and you must re-login.

Authentication Requirements by Endpoint

API Section	Authentication Required
ESG Unified (`/api/v1/esg/`)	Required for most endpoints
OpenAQ (`/api/v1/data-sources/openaq/`)	Required
Climate TRACE (`/api/v1/data-sources/climatetrace/`)	Required
EDGAR (`/api/v1/data-sources/edgar/`)	Required
GLEIF (`/api/v1/data-sources/gleif/`)	Required
GCP (`/api/v1/data-sources/gcp/`)	Required
NOAA (`/api/v1/data-sources/noaa_storm_events/`)	Required
Health checks (`/health/`)	Public
Auth endpoints (`/api/auth/device-code/`, `/api/auth/device-token/`)	Public

Python SDK (Eko Client)

The `jana-eko-client` Python library provides a high-level interface to all API endpoints with automatic authentication, pagination, and retry handling.

Installation

```
# Requires Python 3.10 or 3.11
pip install "git+https://github.com/Jana-Earth-Data/jana-eko-client.git@development"
```

For Jupyter notebooks, see INSTALL.md for venv and kernel setup.

Quick Start

```
from eko_client import EkoUserClient

# Initialize with dual-URL support
```

```
client = EkoUserClient(
    base_url="https://auth-dev.jana.earth",
    api_base_url="https://api-dev.jana.earth",
    client_id="jana-sdk",
    timeout=120,
)

# Authenticate (opens browser)
client.login_device()

# ESG unified data
data = client.get_data(sources="openaq", country_codes="USA", limit=10)

# OpenAQ locations
locations = client.get_openaq_locations(country_codes="US", limit=5)

# Climate TRACE emissions (country_code is alpha-3; use page_size, not limit:
# /emissions/ silently ignores `limit` and returns 5,000 rows by default)
emissions = client.get_climatetrace_emissions(country_code="USA", page_size=10)

# EDGAR country totals
totals = client.get_edgar_country_totals(country_code="USA", year=2022)

# EDGAR grid emissions (bbox required for spatial filtering)
grid = client.get_edgar_grid_emissions(year=2022, gas="CO2", bbox="75,22,95,35")

# Platform summary
summary = client.get_summary()
```

Pagination Helper

Most list endpoints return paginated results. The SDK provides `fetch_all_pages()` to iterate through all pages automatically:

```
# Fetch all EDGAR country totals (cursor pagination, 5000/page)
all_records = client.fetch_all_pages(
    '/api/v1/data-sources/edgar/country-totals/',
    page_size=5000,
    progress=True,   # print progress
    max_pages=100,    # safety limit
)
print(f"Fetched {len(all_records):,} records")
```

This handles both cursor-based and page-number pagination transparently.

Complete Method Reference

Authentication

Method	Description
`login_device()`	OAuth 2.0 device-code flow (opens browser)
`login_password(email, password)`	Username/password login
`refresh_jwt()`	Refresh access token
`get_user_info()`	Get authenticated user profile
`logout()`	Clear stored tokens
`is_authenticated()`	Check if access token exists

ESG Unified Data

Method	Description
`get_data(sources, country_codes, ...)`	Cross-source environmental data
`get_aggregations(temporal_resolution, ...)`	Temporal aggregations
`get_correlations(sources, country_codes, ...)`	Cross-source correlations
`get_trends(parameters, temporal_resolution, ...)`	Trend analysis
`get_quality(sources, ...)`	Data quality insights
`get_alerts(alert_types, severity, ...)`	System alerts
`get_geojson(sources, location_bbox, ...)`	GeoJSON export
`get_locations(sources, country_codes, ...)`	Combined locations
`get_sectors(sources, country_codes, ...)`	Available sectors
`get_summary()`	Platform overview
`get_health()`	ESG service health
`get_system_health()`	Full system health

Metadata & Definitions

Method	Description
`get_definitions()`	Definition categories
`get_parameter_definitions(sources, ...)`	Parameter metadata
`get_unit_definitions()`	Unit definitions
`get_source_definitions()`	Data source descriptions

Data Export

Method	Description
`create_export(format, query, ...)`	Create async export job
`get_export_status(export_id)`	Poll export progress
`download_export(export_id)`	Download completed export

OpenAQ (Air Quality)

Method	Key Parameters	Description
`get_openaq_locations(...)`	`country_codes`, `location_bbox`, `page_size`	Monitoring stations
`get_openaq_location(location_id)`	`location_id`	Single station detail
`get_openaq_sensors(...)`	`location_id`, `parameter`	Sensors at a station
`get_openaq_sensor(sensor_id)`	`sensor_id`	Single sensor detail
`get_openaq_measurements(...)`	`location_id`, `parameter`, `date_from`, `date_to`	Measurement data
`get_openaq_measurement(id)`	`measurement_id`	Single measurement
`get_openaq_measurements_totals()`	--	Record count (MV-backed)
`get_openaq_measurements_parameter_totals()`	--	Totals by parameter (MV — 38 (parameter, unit) rows)

Method	Key Parameters	Description
`get_openaq_measurements_country_totals()`	--	Totals by country (MV — 157 countries; may lag live data)
`get_openaq_measurements_date_range()`	--	Earliest/latest timestamp (MV; country-filtered variant currently times out)
`get_openaq_stats()`	--	Platform summary: station/sensor/measurement/parameter/country counts (MV)
`get_openaq_parameters(...)`	`limit`	Available parameters
`get_openaq_parameter(id)`	`parameter_id`	Single parameter

Climate TRACE (Emissions)

Filter vocabulary on Climate TRACE differs from the REST endpoints -- the Python SDK passes parameters straight through, so the same endpoint-specific rules apply. See the REST "Filter vocabulary" table in the Climate TRACE section below for the authoritative works-vs-silently-ignored breakdown per endpoint. Use alpha-3 country codes (`country_code="USA"`, `"NPL"`, `"CHN"`) on `/emissions/*` and `/assets/*`; use `country_iso3` / `gas_type` / `year` on `/annual-country-emissions/` only.

Method	Key Parameters	Description
`get_climatetrace_sectors(...)`	`limit`	All sectors (26 coarse rows)
`get_climatetrace_countries(...)`	`limit`	All countries (251 rows)
`get_climatetrace_assets(...)`	`sector_id`, `country_code` (alpha-3), `location_bbox`	Emitting assets. `country_iso3` and `sector` (slug) are silently ignored -- use `country_code` and `sector_id`.
`get_climatetrace_emissions(...)`	`asset_id`, `country_code`, `gas`, `date_from`, `date_to`	Emission records. `sector_name`, `country_iso3`, `gas_type`, `year` are silently ignored.
`get_climatetrace_emissions_totals(...)`	`country_code`, `gas`, `date_from`, `date_to`	Aggregate totals (freshest MV)
`get_climatetrace_emissions_sector_totals(...)`	`country_code`, `gas`	Totals by sector (served from stale MV -- 73k-row gap)
`get_climatetrace_emissions_country_totals(...)`	`gas`	Totals by country (stale MV; `country_name` duplicates `country_iso3`; `unique_assets=0`)
`get_climatetrace_emissions_gas_type_distribution(...)`	`country_code`	Gas breakdown (only `co2e` is populated; CO2/CH4/N2O rows do not exist

Method	Key Parameters	Description
		in the emissions table)
`get_climatetrace_annual_country_emissions(...)`	`country_iso3`, `gas_type` (`co2/ch4/n2o/co2e_100yr/co2e_20yr`), `year`	**Only** source of per-gas CO2/CH4/N2O splits. Inverted filter vocabulary vs `/emissions/*`.
`get_climatetrace_company_matches(...)`	`asset_id`	Company-asset ownership links (6,227 rows)
`get_climatetrace_violations(...)`	`asset_id`	Environmental violations. **Currently empty.**

SDK caveats for Climate TRACE:

- The SDK wrapper for `/assets/aggregated-emissions/` returns CamelCase keys (`AssetCount`, `Emissions`, `Year`, `Month`, `Gas`, `Sector`) -- the only CT endpoint that does so. Plan your dict-access accordingly.
- `/sectors/{id}/emissions_summary/` returns `null` CO2e aggregates despite a populated asset count; prefer `sector_totals` with `sector_id`.
- Cursor endpoints do not return a `count` field; use the `_totals` helpers if you need record counts.

EDGAR (Global Emissions)

Method	Key Parameters	Description
`get_edgar_country_totals(...)`	`country_code`, `year`, `gas`, `sector`	National totals
`get_edgar_grid_emissions(...)`	`year`, `gas`, `bbox`, `coordinates`, `radius`	Gridded emissions (0.1 degree)
`get_edgar_temporal_profiles(...)`	`sector`, `temporal_level`, `month`	Monthly/hourly factors
`get_edgar_fasttrack(...)`	`country_code`, `year`, `gas`	Provisional annual data

All methods have async variants (append `_async` to the method name).

Common Concepts

Pagination

Two pagination styles are used across the API:

Cursor-based pagination (EDGAR, Climate TRACE bulk endpoints):

Used for large datasets. Returns an opaque `next` URL -- no `count` field. Page sizes up to 5,000 (max 10,000 for EDGAR).

```
{
  "next": "https://api-dev.jana.earth/api/v1/data-sources/edgar/country-totals/?cursor=cD0yMDI0...",
  "previous": null,
  "results": [...]
}
```

Page-number pagination (OpenAQ, ESG endpoints):

```
{
  "count": 5432,
  "next": "https://api-dev.jana.earth/api/v1/data-sources/openaq/locations/?page=2",
  "previous": null,
```

```
  "results": [...]
}
```

Parameter	Type	Default	Max	Description
`page`	integer	1	--	Page number (page-number only)
`page_size`	integer	100	varies	Results per page
`cursor`	string	--	--	Opaque cursor token (cursor only)

Filtering

Most endpoints support:

- **Filter fields**: Exact match (e.g., `?country_code=USA`)
- **Search**: Full-text via `?search=keyword`
- **Ordering**: Sort via `?ordering=field` or `?ordering=-field` (descending)

Geographic Parameters

Parameter	Format	Example	Description
`country_codes`	ISO 3166-1 alpha-2 or alpha-3	`USA,CAN,MEX`	Comma-separated country codes
`bbox` / `location_bbox`	`min_lon,min_lat,max_lon,max_lat`	`-74.1,40.6,-73.9,40.8`	Bounding box
`coordinates` / `location_point`	`lon,lat`	`-73.935,40.730`	Point coordinates
`radius` / `radius_km`	float (km)	`25`	Radius for point-based search

Date Parameters

Parameter	Format	Example
`date_from`	ISO 8601	`2024-01-01T00:00:00Z`
`date_to`	ISO 8601	`2024-12-31T23:59:59Z`

ESG Unified API

The ESG Unified API (`/api/v1/esg/`) provides cross-source data access, analytics, and export capabilities.

Unified Data

`GET /api/v1/esg/data/`

The primary endpoint for cross-source environmental data queries.

Parameter	Type	Required	Default	Description
`country_codes`	string	**Yes**	--	ISO country codes (comma-separated)
`sources`	string	No	all	`openaq`, `climatetrace`, `edgar`
`location_bbox`	string	No	--	`min_lon,min_lat,max_lon,max_lat`
`location_point`	string	No	--	`lon,lat`
`radius_km`	float	No	10	Radius in km for point search
`date_from`	datetime	No	--	Start date (ISO 8601)
`date_to`	datetime	No	--	End date (ISO 8601)
`parameters`	string	No	--	Environmental parameters (e.g., `pm25,co2`)
`include_correlations`	boolean	No	false	Include cross-source correlations
`limit`	integer	No	1000	Max results (max: 10,000)

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/esg/data/?sources=openaq,climatetrace&country_codes=USA&limit=5"
```

Parameters

`GET /api/v1/esg/parameters/` -- All available environmental parameters across sources.

Sectors

`GET /api/v1/esg/sectors/` -- Sectors with emissions data. Requires `country_codes`.

Locations

`GET /api/v1/esg/locations/` -- Combined location data from all sources. Requires `country_codes`.

Aggregations

`GET /api/v1/esg/aggregations/` -- Pre-computed temporal aggregations. Requires `country_codes` and `temporal_resolution` (`hourly`, `daily`, `monthly`).

Correlations

`GET /api/v1/esg/correlations/` -- Cross-source correlation analysis. Requires `country_codes` and at least 2 `sources`.

Trends

`GET /api/v1/esg/trends/` -- Temporal trend analysis. Supports `daily`, `weekly`, `monthly`, `yearly` resolution.

Quality Insights

`GET /api/v1/esg/quality/` -- Comprehensive data quality analysis.

Platform Summary

`GET /api/v1/esg/summary/` -- High-level platform overview with record counts, freshness, and health.

Analytics

`GET /api/v1/esg/analytics/` -- Advanced cross-source correlation analytics.

System Health

`GET /api/v1/esg/system-health/` -- Full system health with ingestion status, API performance, and database health.

Statistics

`GET /api/v1/esg/statistics/` -- List all tables with record counts and freshness.

`GET /api/v1/esg/statistics/{table_name}/` -- Detailed statistics for a specific table.

`GET /api/v1/esg/openaq-statistics/` -- OpenAQ-specific summary.

Definitions

`GET /api/v1/esg/definitions/` -- Definition categories.

`GET /api/v1/esg/definitions/units/` -- Unit definitions and conversion formulas.

`GET /api/v1/esg/definitions/sources/` -- Data source descriptions and coverage.

Data Export (Synchronous)

`GET /api/v1/esg/export/` -- Download data as JSON or CSV.

Parameter	Type	Required	Default	Description
`table_name`	string	**Yes**	--	Table to export
`export_format`	string	No	`json`	`json` or `csv`
`limit`	integer	No	10,000	Max records (max: 100,000)

Data Export (Async)

For large datasets:

1. **Create**: `POST /api/v1/esg/exports/` with `query_params` and `export_format`

2. **Poll**: `GET /api/v1/esg/exports/{job_id}/` until `status: completed`
3. **Download**: `GET /api/v1/esg/exports/{job_id}/download/` (redirects to S3)

Source-Specific APIs

OpenAQ -- Air Quality Data

Base path: `/api/v1/data-sources/openaq/`

Authentication required for all endpoints.

Inventory (live as of 2026-04-22): 50,514 monitoring stations (a.k.a. "locations") across 159 countries, 311,602 sensors, ~1,489,718,000 measurements, 30 parameters. Coverage 2016-01-30 → present.

Terminology: A *location* is a monitoring station. Each station typically hosts multiple *sensors*, one per parameter (PM2.5, PM10, O3, temperature, …). Query locations when you want geography; query sensors when you want per-parameter coverage.

Endpoints

Endpoint	Method	Description
`/parameters/`	GET	Available air quality parameters
`/locations/`	GET	Monitoring stations (filterable by country, bbox, point+radius)
`/locations/{id}/`	GET	Single station detail
`/locations/{id}/sensors/`	GET	Sensors at a station
`/sensors/`	GET	All sensors (filterable)
`/sensors/{id}/`	GET	Single sensor detail
`/measurements/`	GET	Measurement data (use `date_from`/`date_to` to widen beyond the default 7-day window)
`/measurements/{id}/`	GET	Single measurement
`/measurements/totals/`	GET	Record count
`/measurements/parameter_totals/`	GET	Totals by parameter (returns 38 `(parameter, unit)` rows)
`/measurements/country_totals/`	GET	Totals by country (**materialized-view backed — may lag live data**, see Data Freshness note below)
`/measurements/date_range/`	GET	Earliest/latest timestamps (**unfiltered only**; `?location__country_code=X` currently times out — use the workaround in the Recipes section)
`/stats/`	GET	Summary statistics (MV-backed — may lag live data)

Key filters — important naming

Filter	Locations	Sensors	Measurements
Country, single-code, ISO-2 (e.g. NP)	`country_code=NP`	**`location__country_code=NP`** ⚠	**`location__country_code=NP`** ⚠
Country, multi-code (e.g. `US,CA`)	`country_codes=US,CA`	—	—
Reference-grade only	`is_monitor=true`	—	—
Bounding box	`bbox=…`	`location_bbox=…`	`coordinates=lon,lat&radius=km`
Parameter	—	`parameter=pm25`	`parameters=pm25`
Date window	—	—	`date_from=…`, `date_to=…` (ISO-8601)

⚠ **Important — `country_code` vs `location__country_code`:** On `/sensors/` and `/measurements/`, the correct filter key is `location__country_code` (double underscore, Django ORM traversal syntax). Passing `?

`country_code=NP` is **silently ignored** and returns all ~310K global sensors instead of the ~830 Nepal sensors. Always use `location__country_code` for these two endpoints. This will be simplified in a future release; for now the double-underscore form is required.

Recipes

```
# California reference-grade monitoring stations
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/openaq/locations/?country_codes=US&is_monitor=true&bbox=-124.4,32

# PM2.5 measurements near Tokyo (last 3 months)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/openaq/measurements/?coordinates=139.69,35.68&radius=30&parameter

# Count Nepal sensors (correct filter form)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/openaq/sensors/?location__country_code=NP&page_size=1"
# -> {"count": 830, ...}

# Country-scoped date range (workaround until /date_range/?location__country_code=X is fixed)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/openaq/measurements/?location__country_code=NP&date_from=2000-01-
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/openaq/measurements/?location__country_code=NP&date_from=2000-01-
```

Data freshness & caveats

- **MV-backed aggregates may lag live data.** `/stats/`, `/measurements/country_totals/`, `/measurements/parameter_totals/`, and `/measurements/date_range/` are served from materialized views refreshed on a fixed cadence. Exact live counts (and the most recent measurements) come from `/measurements/?…&page_size=1&ordering=±measured_at`. If an aggregate and a live query disagree, the live query is authoritative.
- **Counts use `(parameter_name, unit)` pairs.** `/measurements/parameter_totals/` returns 38 rows while `/stats/` reports 30 distinct parameter *names*; the difference is that some parameters (e.g. PM2.5) have been reported in more than one unit across the network's history.
- **Parameter availability is country-specific.** The global headline list (PM2.5, PM10, NO2, O3, CO, SO2) is not what every country reports. Example: Nepal's 121 stations / 830 sensors cover 12 parameters — primarily PM2.5, PM10, PM1, O3, temperature, relative humidity, and particle-count channels (`um003`, `um010`, `um025`, `um100`, in particles/cm³). There are no NO2, SO2, or CO sensors in Nepal. Use `GET /sensors/?location__country_code=X` to enumerate what is actually reported for a country.
- **Raw-feed outliers exist.** OpenAQ values are ingested without sanity bounds. A small number of records contain physically implausible values (e.g. NO2 > 1,000,000 µg/m³, negative PM). Aggregates on `/parameter_totals/` reflect the raw feed; apply your own bounds before use.

Climate TRACE -- Emissions Data

Base path: `/api/v1/data-sources/climatetrace/`

Authentication required. Uses cursor-based pagination for `/assets/` and `/emissions/`; page-number pagination elsewhere.

Inventory (live as of 2026-04): 1,337,615 "emitting assets" across 45+ sector slugs and 251 countries; 53,495,196 emission records (freshest materialized view) covering recent history through 2024.

Climate TRACE uses the term **"asset"** for any discrete emitter, which spans two very different categories:

- **~153,000 named industrial facilities** -- discrete plant-like emitters you can identify by name and operator: power plants (9,732), manufacturing sites (33,685), waste sites (96,099, including landfills and WWTPs), plus fossil fuel operations, mining, and fluorinated-gas plants.
- **~1,184,000 gridded/polygon assets** -- spatial units, not discrete plants: agriculture 1,002,010 (field/farm polygons), buildings 114,070 (1 km² grid tiles), transportation 81,918 (shipping tracks, road segments, airports).

The sum is 1,337,615. Earlier docs quoted "70,000+ facilities" -- that referred only to the plant-like-facility subset at CT launch (2022) and is superseded by the current database ground truth above. **When sizing your use case, be explicit about which category you need:** a "facilities" use case typically maps to the ~153K named industrial subset; a "coverage" or "gridded emissions" use case maps to the full 1.34M.

Endpoints

Endpoint	Method	Pagination	Description
`/sectors/`	GET	Page	Emission sectors (26 coarse sectors; asset-level sector slugs are finer-grained, 45+)
`/countries/`	GET	Page	Countries with data (251 rows)
`/assets/`	GET	Cursor	Emitting assets (1,337,615 total; ~153K named industrial facilities + ~1.18M gridded/polygon assets — see Inventory note above)
`/assets/{id}/`	GET	--	Single asset detail
`/assets/{id}/emissions/`	GET	Page	Asset emission history
`/assets/{id}/violations/`	GET	Page	Per-asset violations (currently empty)
`/assets/aggregated-emissions/`	GET	--	Per-sector aggregates. **Returns CamelCase keys** (`AssetCount`, `Emissions`, `Year`, `Month`, `Gas`, `Sector`) -- unlike every other CT endpoint.
`/emissions/`	GET	Cursor	All emission records
`/emissions/totals/`	GET	--	Aggregate totals (freshest MV)
`/emissions/sector_totals/`	GET	--	Totals by sector (7 MV rows)
`/emissions/country_totals/`	GET	--	Totals by country (252 rows; one "unknown" bucket beyond `/countries/`)
`/emissions/gas_type_distribution/`	GET	--	Gas breakdown (see "Gas column semantics" caveat below)
`/emissions/date_range/`	GET	--	Min/max date across CT emissions
`/countries/{id}/assets/`	GET	Page	Assets in a given country (nested)
`/sectors/{id}/assets/`	GET	Page	Assets in a given sector (nested)
`/sectors/{id}/emissions_summary/`	GET	--	**Known issue:** `total_assets` populates but `total_co2e_tonnes` / `avg_co2e_tonnes` return `null`. Use `/emissions/sector_totals/` instead until fixed.
`/company-matches/`	GET	Page	Company-asset ownership links (6,227 rows)
`/violations/`	GET	Page	Environmental-violation records. **Currently empty (count = 0)** -- Climate TRACE upstream has not published violations data on Jana.
`/annual-country-emissions/`	GET	Page	Annual country-level totals with per-gas breakdown

Filter vocabulary is endpoint-specific on Climate TRACE. Three Climate TRACE endpoints currently accept *different* filter parameter names for the same underlying concept, and unknown parameters are silently ignored (DRF returns the unfiltered result set with no warning). Use the table below until ticket #141 / #142 / #143 land.

Endpoint	Works	Silently ignored
`/emissions/` + `/emissions/totals/` + `/emissions/*_totals/` + `/emissions/gas_type_distribution/`	`country_code` (alpha-3), `gas`, `date_from`, `date_to`, `temporal_granularity`, `asset_id`	`country_iso3`, `country_name`, `gas_type`, `sector`, `sector_name`, `year`
`/annual-country-emissions/`	`country_iso3`, `gas_type`, `year`	`country_code`, `gas`, `date_from`, `date_to`

Endpoint	Works	Silently ignored
`/assets/`	`country_code` (alpha-3), `sector_id`, `location_bbox`, `coordinates+radius`, `ordering`	`country_iso3`, `sector`, `sector_name`

Canonical alpha-3 codes are used throughout (`USA`, `CHN`, `NPL`, `IND`). Alpha-2 codes are **not** resolved on Climate TRACE endpoints.

Gas column semantics (important). On `climatetrace_emissions` (and therefore `/emissions/`, `/emissions/totals/`, `/emissions/country_totals/`, `/emissions/sector_totals/`, `/emissions/gas_type_distribution/`), the `gas` column is effectively `co2e` for 99.999998% of rows (one `co2e_100yr` row exists; `co2`, `ch4`, and `n2o` rows do **not** exist in the denormalized emissions table). The `total_co2`, `total_ch4`, and `total_n2o` fields on `/emissions/totals/` are always `0.0` for the same reason.

Per-gas CO2 / CH4 / N2O splits are only available via `/annual-country-emissions/?gas_type={co2|ch4|n2o|co2e_100yr|co2e_20yr}`, which operates on a separate backend table.

Pagination asymmetry on cursor endpoints.

- `/emissions/` uses `BulkCursorPagination` with a 5,000-row default. `?page_size=N` is honoured; `?limit=N` is **silently ignored** (you will receive 5,000 rows). Always use `page_size` on `/emissions/`.
- `/assets/` uses a separate cursor class and accepts both `page_size` and `limit` with a smaller default.
- Neither cursor endpoint returns a `count` field; use `/emissions/totals/` to derive record counts.

Materialized-view freshness. `/emissions/totals/` reflects the live / freshest aggregate (53,495,196 rows). `/emissions/country_totals/`, `/emissions/sector_totals/`, `/emissions/gas_type_distribution/`, and `/emissions/date_range/` are served from an older shared MV currently 73,004 rows behind (53,422,192). Treat the MV-backed endpoints as snapshots, not real-time.

Known data-quality caveats.

- `unique_assets` on `/emissions/country_totals/` and `/emissions/sector_totals/` is always `0` (not populated).
- `country_name` on `/emissions/country_totals/` currently duplicates `country_iso3` -- it is **not** the human-readable country name.
- `/sectors/{id}/emissions_summary/` returns `null` for `total_co2e_tonnes` and `avg_co2e_tonnes`. Use `/emissions/sector_totals/?sector_id=N` instead.

```
# Top emitting assets in India (use alpha-3 + sector_id, not country_iso3 / sector)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/climatetrace/assets/?country_code=IND&ordering=-latest_co2e_tonne

# Power-sector emissions by country (co2e is the only gas with row-level data)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/climatetrace/emissions/sector_totals/?gas=co2e"

# Annual per-gas country totals (separate vocabulary: country_iso3 + gas_type + year)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/climatetrace/annual-country-emissions/?country_iso3=NPL&gas_type=
```

EDGAR -- Global Emissions Modeling

Base path: `/api/v1/data-sources/edgar/`

Authentication required. All EDGAR endpoints use **cursor-based pagination** (BulkCursorPagination: 5,000 records/page, max 10,000).

Endpoints

Endpoint	Method	Description
`/country-totals/`	GET	National emissions by year, gas, sector
`/grid-emissions/`	GET	0.1 degree gridded emissions (at least one filter required)

Endpoint	Method	Description
`/grid-emissions/summary/`	GET	Precomputed grid summary (from materialized view)
`/temporal-profiles/`	GET	Monthly/hourly disaggregation factors
`/fasttrack/`	GET	Provisional annual emissions
`/air-pollutant-totals/`	GET	Air pollutant country totals
`/air-pollutant-grid/`	GET	Air pollutant gridded data (at least one filter required)

Country Totals

Filters: `country_code`, `year`, `gas_type`, `sector`, `provisional`

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/edgar/country-totals/?country_code=USA&year=2022&gas_type=CO2&pag
```

Note: In the `EDGAR_2024_GHG` dataset, the `sector` field contains **world region groupings** (e.g., "China +", "OECD_Europe"), not EDGAR activity sectors.

Grid Emissions

Filters: `year`, `gas`, `sector`, `bbox`, `coordinates`, `radius`

At least one filter is required to prevent unfiltered scans across millions of grid cells.

```
# Nepal region, CO2, 2022
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/edgar/grid-emissions/?year=2022&gas=CO2&bbox=75,22,95,35&page_siz
```

Spatial filters:

- `bbox=min_lon,min_lat,max_lon,max_lat` -- Bounding box (PostGIS `within`)
- `coordinates=lon,lat` + `radius=km` -- Point + radius (PostGIS `distance_lte`)

Temporal Profiles

Filters: `sector`, `temporal_level` (`monthly`, `hourly`), `month`, `hour`, `profile_version`

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/edgar/temporal-profiles/?temporal_level=monthly&page_size=5000"
```

FastTrack

Filters: `country_code`, `year`, `gas`, `sector`, `provisional`

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/edgar/fasttrack/?country_code=NPL&year=2022"
```

GLEIF -- Legal Entity Identifiers

Base path: `/api/v1/data-sources/gleif/`

Authentication required. Uses page-number pagination.

GLEIF provides the **Legal Entity Identifier (LEI)** system -- a global, standardized 20-character alphanumeric identifier (ISO 17442) for legal entities in financial transactions. The Jana API exposes three GLEIF datasets: entities (3.25M+), corporate ownership relationships (637K+), and reporting exceptions (5.8M+).

Endpoints

Endpoint	Method	Description
`/entities/`	GET	Legal entities -- search by name, LEI, BIC; filter by country/status/category
`/entities/{lei}/`	GET	Full entity detail by LEI (20-char alphanumeric, e.g. `HWUPKR0MPOU8FGXBT394`)
`/entities/{lei}/parents/`	GET	Direct and ultimate parent entities
`/entities/{lei}/children/`	GET	Direct child entities (subsidiaries)
`/relationships/`	GET	Entity ownership relationships
`/exceptions/`	GET	Reporting exceptions (why an entity cannot report a parent)

Entity List & Search

Filters: `entity_status`, `registration_status`, `entity_category`, `legal_address_country`, `headquarters_country`, `jurisdiction`

Search fields: `legal_name`, `lei`, `bic`, `registered_as` (via `?search=`)

Ordering: `legal_name`, `last_update_date`, `entity_creation_date`

```
# Search for entities by name
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/entities/?search=Apple+Inc&limit=5"

# Filter US entities in FUND category
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/entities/?legal_address_country=US&entity_category=FUND&lim

# Active entities in Japan
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/entities/?legal_address_country=JP&entity_status=ACTIVE&lim
```

List response fields (compact): `id`, `lei`, `legal_name`, `legal_address_country`, `headquarters_country`, `jurisdiction`, `entity_category`, `entity_status`, `registration_status`

Entity Detail

```
# Apple Inc. (LEI: HWUPKR0MPOU8FGXBT394)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/entities/HWUPKR0MPOU8FGXBT394/"
```

Detail response fields (full): `id`, `lei`, `legal_name`, `legal_name_language`, `other_names`, `legal_address_lines`, `legal_address_city`, `legal_address_region`, `legal_address_country`, `legal_address_postal_code`, `headquarters_city`, `headquarters_country`, `jurisdiction`, `entity_category`, `entity_sub_category`, `legal_form_code`, `legal_form_other`, `entity_status`, `entity_creation_date`, `registration_status`, `initial_registration_date`, `last_update_date`, `next_renewal_date`, `managing_lou`, `bic`, `registered_as`, `successor_lei`, `ingested_at`, `updated_at`

Corporate Hierarchy (Parents & Children)

```
# Get parent entities (direct and ultimate)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/entities/HWUPKR0MPOU8FGXBT394/parents/"
```

Response:

```
{
  "direct_parent": {
    "lei": "...", "legal_name": "...", "legal_address_country": "..."
  },
  "ultimate_parent": {
    "lei": "...", "legal_name": "...", "legal_address_country": "..."
```

```
  }
}
```

Both fields are `null` if the entity has no parent (e.g., Apple Inc. is a top-level entity).

```
# Get child entities (direct subsidiaries)
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/entities/HWUPKR0MPOU8FGXBT394/children/"
```

Returns an array of child entities with the compact list fields.

Relationships

Filters: `start_lei`, `end_lei`, `relationship_type`, `relationship_status`

Relationship types: `IS_DIRECTLY_CONSOLIDATED_BY`, `IS_ULTIMATELY_CONSOLIDATED_BY`, `IS_FEEDER_TO`, `IS_FUND-MANAGED_BY`

```
# All relationships for a specific entity
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/relationships/?start_lei=HWUPKR0MPOU8FGXBT394"
```

Response fields: `id`, `start_lei`, `end_lei`, `relationship_type`, `relationship_status`, `start_date`, `end_date`, `ingested_at`

Reporting Exceptions

Filters: `lei`, `exception_category`, `exception_reason`

Exception categories: `DIRECT_ACCOUNTING_CONSOLIDATION_PARENT`, `ULTIMATE_ACCOUNTING_CONSOLIDATION_PARENT`

Exception reasons: `NON_CONSOLIDATING`, `NO_KNOWN_PERSON`, `NO_LEI`, `NATURAL_PERSONS`, `NON_PUBLIC`, `BINDING_LEGAL_OBSTACLES`, `CONSENT_NOT_OBTAINED`, `DETRIMENT_NOT_EXCLUDED`, `DISCLOSURE_DETRIMENTAL`

```
# Exceptions for a specific entity
curl -H "Authorization: Bearer $TOKEN" \
  "https://api-dev.jana.earth/api/v1/data-sources/gleif/exceptions/?lei=HWUPKR0MPOU8FGXBT394"
```

Response fields: `id`, `lei`, `exception_category`, `exception_reason`, `ingested_at`

GCP -- Global Carbon Project

Base path: `/api/v1/data-sources/gcp/`

Authentication required. Standard page-number pagination.

The Global Carbon Project provides national CO2 emissions, fossil fuel breakdowns, and global carbon/methane budgets spanning 1750-2023.

National Emissions (`/national-emissions/`)

National territorial and consumption-based CO2 emissions by country and year, including per-capita values.

Parameter	Type	Description
`country_code`	string	ISO-3 country code (e.g. `USA`, `CHN`, `IND`)
`year`	int	Emission year (e.g. `2020`)
`budget_version`	string	GCP budget version (e.g. `2024`)
`limit`	int	Results per page

```
# USA emissions for 2020
emissions = client.get_gcp_national_emissions(country_code="USA", year=2020)
```

```
# curl
curl -H "Authorization: Bearer $JWT" \
  "https://api-test.jana.earth/api/v1/data-sources/gcp/national-emissions/?country_code=USA&year=2020&limit=5"
```

Response fields: `id`, `country_code`, `country_name`, `year`, `territorial_mtco2`, `consumption_mtco2`, `transfer_mtco2`, `per_capita_tco2`, `data_source_name`, `budget_version`, `created_at`, `updated_at`

Emissions by Fuel (`/emissions-by-fuel/`)

Global fossil CO2 emissions broken down by fuel type (coal, oil, gas, cement, flaring, other).

Parameter	Type	Description
`year`	int	Emission year (e.g. `2020`)
`budget_version`	string	GCP budget version (e.g. `2024`)
`limit`	int	Results per page

```
# Fuel mix for 2020
fuel_data = client.get_gcp_emissions_by_fuel(year=2020)
```

```
curl -H "Authorization: Bearer $JWT" \
  "https://api-test.jana.earth/api/v1/data-sources/gcp/emissions-by-fuel/?year=2020&limit=5"
```

Response fields: `id`, `year`, `coal_gtco2`, `oil_gtco2`, `gas_gtco2`, `cement_gtco2`, `flaring_gtco2`, `other_gtco2`, `total_gtco2`, `budget_version`, `created_at`, `updated_at`

Carbon Budget (`/carbon-budget/`)

Global carbon budget components: fossil emissions, land-use change, atmospheric growth, ocean sink, land sink.

Parameter	Type	Description
`year`	int	Budget year (e.g. `2020`)
`budget_version`	string	GCP budget version (e.g. `2024`)
`limit`	int	Results per page

```
budget = client.get_gcp_carbon_budget(year=2020)
```

Note: Carbon budget data has not yet been ingested. This endpoint is available but currently returns empty results.

Methane Budget (`/methane-budget/`)

Global methane budget components: anthropogenic and natural sources and sinks.

Parameter	Type	Description
`year`	int	Budget year (e.g. `2020`)
`budget_version`	string	GCP budget version (e.g. `2024`)
`limit`	int	Results per page

```
methane = client.get_gcp_methane_budget(year=2020)
```

Note: Methane budget data has not yet been ingested. This endpoint is available but currently returns empty results.

NOAA -- Storm Events

Base path: `/api/v1/data-sources/noaa_storm_events/`

Authentication required. Standard page-number pagination.

NOAA Storm Events provides detailed records of severe weather events across the United States (2015-2024), including tornadoes, hurricanes, floods, hail, thunderstorms, winter storms, and more.

Storm Events (`/events/`)

Parameter	Type	Description
`event_type`	string	Event type (e.g. `Tornado`, `Hurricane`, `Flood`, `Hail`, `Thunderstorm Wind`, `Winter Storm`, `Blizzard`, `Ice Storm`, `Drought`, `Wildfire`)
`state`	string	US state in uppercase (e.g. `TEXAS`, `FLORIDA`, `CALIFORNIA`)
`year`	int	Event year (2015-2024)
`month`	int	Event month (1-12)
`bbox`	string	Bounding box as `min_lat,min_lon,max_lat,max_lon`
`lat`	float	Latitude for point+radius search (-90 to 90)
`lon`	float	Longitude for point+radius search (-180 to 180)
`radius_km`	float	Search radius in km (requires `lat` and `lon`)
`limit`	int	Results per page

```
# Texas tornadoes in January 2023
storms = client.get_noaa_storm_events(
    event_type="Tornado", state="TEXAS", year=2023, month=1, limit=100
)

# Tornadoes near Houston (50km radius)
storms = client.get_noaa_storm_events(
    lat=29.76, lon=-95.37, radius_km=50, event_type="Tornado", year=2023
)
```

```
curl -H "Authorization: Bearer $JWT" \
  "https://api-test.jana.earth/api/v1/data-sources/noaa_storm_events/events/?event_type=Tornado&state=TEXAS&year=20
```

Response fields: `id`, `event_id`, `episode_id`, `event_type`, `state`, `state_fips`, `cz_type`, `cz_fips`, `cz_name`, `begin_date`, `end_date`, `injuries_direct`, `injuries_indirect`, `deaths_direct`, `deaths_indirect`, `total_injuries`, `total_deaths`, `damage_property`, `damage_crops`, `source`, `magnitude`, `magnitude_type`, `tor_f_scale`, `year`, `month`, `latitude`, `longitude`, `geometry`, `created_at`, `updated_at`

Real-World Use Cases

1. Nepal Air Quality + Emissions Correlation

Nepal inventory at time of writing: **121 monitoring stations, 830 sensors, ~1.88M measurements spanning 2017-03-02 → 2026-04-10, 12 parameters** (primarily PM2.5/PM10/PM1, O3, temperature/RH, particle counts).

```
client.login_device()

# Air quality stations in Nepal (uses `country_codes` — plural, ISO-2)
locations = client.get_openaq_locations(country_codes="NP", limit=100)

# Sensors in Nepal — NOTE: use `location__country_code` for sensors/measurements
# (ORM traversal key; see "Important — country_code vs location__country_code" above)
sensors = client.get_openaq_sensors(location__country_code="NP", page_size=500)

# Recent Nepal measurements
measurements = client.get_openaq_measurements(
    location__country_code="NP",
    date_from="2026-01-01",
    page_size=1000,
```

```
)

# EDGAR gridded emissions over Nepal
grid = client.get_edgar_grid_emissions(
    year=2022, gas="CO2",
    bbox="80,26,89,31",
    limit=10000
)

# Climate TRACE facilities in Nepal -- country_code is alpha-3 on
# /assets/ (NOT alpha-2, NOT "country_iso3"). Both the REST name
# country_iso3 and the alpha-2 form "NP" are silently ignored.
assets = client.get_climatetrace_assets(country_code="NPL")
```

2. Country-Level GHG Comparison

```
# EDGAR country totals for top emitters
for country in ['CHN', 'USA', 'IND', 'RUS', 'JPN']:
    totals = client.get_edgar_country_totals(country_code=country, year=2022, gas="CO2")
    print(f"{country}: {len(totals.get('results', []))} records")

# Climate TRACE country totals -- served from a shared MV that is
# currently ~73k rows behind /emissions/totals/. `unique_assets` is
# always 0 and `country_name` duplicates `country_iso3` here. For the
# freshest totals, call get_climatetrace_emissions_totals() per country.
ct_totals = client.get_climatetrace_emissions_country_totals(gas="co2e")

# Per-gas CO2/CH4/N2O splits (only source -- filter vocabulary is
# inverted vs /emissions/*: country_iso3, gas_type, year all work here)
ct_annual = client.get_climatetrace_annual_country_emissions(
    country_iso3="NPL", gas_type="co2", year=2022
)
```

3. Facility Due Diligence

```
# Find power plants near a location
assets = client.get_climatetrace_assets(
    sector_id=1,  # Power
    location_bbox="-118.7,33.7,-117.6,34.3",
)

# Get emission history for a specific facility
for asset in assets.get('results', [])[:3]:
    history = client.get_climatetrace_emissions(asset_id=asset['id'])
    print(f"{asset['asset_name']}: {len(history.get('results', []))} emission records")
```

4. Corporate Hierarchy + Emissions Cross-Reference

```
# Find a company by name
entities = client.get_gleif_entities(search="Alphabet Inc", limit=5)
alphabet = entities['results'][0]
print(f"{alphabet['legal_name']} (LEI: {alphabet['lei']})")

# Get subsidiaries
children = client.get_gleif_entity_children(alphabet['lei'])
print(f"Subsidiaries: {len(children)}")

# For each subsidiary, check for Climate TRACE asset matches
for child in children[:5]:
    matches = client.get_climatetrace_company_matches(legal_entity_lei=child['lei'])
```

```
    if matches.get('results'):
        print(f"  {child['legal_name']}: {len(matches['results'])} asset matches")
```

5. Bulk Data Export

```
# Create async export
job = client.create_export(
    format="csv",
    query={"sources": ["openaq"], "country_codes": ["USA", "CAN"]},
)
print(f"Job ID: {job['job_id']}")

# Poll until complete
import time
while True:
    status = client.get_export_status(job['job_id'])
    if status['status'] == 'completed':
        break
    time.sleep(10)

# Download
data = client.download_export(job['job_id'])
```

Error Handling

HTTP Status Codes

Code	Description
`200`	Success
`202`	Accepted (async export created)
`302`	Redirect (export download to S3)
`400`	Bad Request -- invalid parameters
`401`	Unauthorized -- missing, invalid, or expired token
`403`	Forbidden -- insufficient permissions (or expired JWT)
`404`	Not Found
`408`	Query timeout (honest -- real data couldn't be computed in time)
`429`	Too Many Requests
`500`	Internal Server Error
`502`	Bad Gateway -- upstream timeout

Error Response Format

```
{
  "error": "Invalid query parameters",
  "details": {
    "country_codes": "This field is required."
  }
}
```

Handling Timeouts

1. **Add spatial filters**: Use `bbox`, `country_code`, or date ranges to narrow scope
2. **Reduce page size**: Try `page_size=100` for complex queries
3. **Use source-specific endpoints**: They are optimized for their data type
4. **Use async export**: For large bulk data needs

Handling Expired Tokens

The Python SDK refreshes JWT tokens automatically. If you see `EkoSessionExpiredError`, your refresh token has expired -- call `client.login_device()` again.

For curl users: POST to `/api/auth/refresh/` with the refresh token to get a new access token.

Rate Limits

Endpoint	Limit
Device code (`/api/auth/device-code/`)	5 requests/minute per IP
Login (`/api/auth/login/`)	5 requests/minute per IP
General API	No hard limit (subject to change)

Attribution Requirements

Source	Attribution
OpenAQ	"Data provided by OpenAQ (openaq.org). Licensed under CC BY 4.0."
Climate TRACE	"Data provided by Climate TRACE (climatetrace.org)."
EDGAR	"Data provided by the European Commission Joint Research Centre (JRC) EDGAR database."
GLEIF	"Data provided by the Global Legal Entity Identifier Foundation (GLEIF). Licensed under CC0 1.0 Universal."
GCP	"Data provided by the Global Carbon Project (globalcarbonproject.org). Licensed under CC BY 4.0."
NOAA	"Data provided by NOAA National Centers for Environmental Information (NCEI) Storm Events Database."

Attribution is included in API responses under the `attribution` field.

Interactive API Documentation

- **Swagger UI**: `https://api-dev.jana.earth/api/docs/`
- **ReDoc**: `https://api-dev.jana.earth/api/redoc/`
- **OpenAPI Schema**: `https://api-dev.jana.earth/api/schema/`

Health Check Endpoints

Endpoint	URL	Auth	Description
Basic health	`GET /health/`	Public	Service liveness
ESG health	`GET /api/v1/esg/health/`	Auth	ESG service status
System health	`GET /api/v1/esg/system-health/`	Auth	Full system diagnostics
Management health	`GET /api/v1/management/health/`	Auth	Management service status

For Management API documentation (job orchestration, ingestion, migrations), see the Jana Management API User Guide.
