Indition Commerce

Indition Commerce API

User Manual — the complete guide to connecting to the Indition Commerce API: concepts, authentication, reading & writing data, scopes, rate limits, errors, security and auditing.

API Module · User Manual (v2) · June 2026

Contents

1. Welcome

Welcome to the **Indition Commerce API**. This manual is the complete, plain-language guide to connecting your software to the platform — reading and updating products, orders, customers and reference data over a secure HTTP API.

It is written for the **integrator**: the developer or technical partner building an integration. You do not need to know the platform's internals — only how to send HTTPS requests and handle JSON. Every example uses `curl` so it is language-neutral.

> New here? Read §1–§4, run the five-minute Quick Start (§4), then dip into the resource and reference sections as you build. The companion **API Reference** lists every endpoint and field; the **Authentication & Getting Started** guide is the short version of §3–§5.

2. Key concepts

Five ideas underpin everything else.

Client

An administrator registers a **client** for your integration. The client *is* your identity — there is no human login. It has a public `client_id`, a confidential `client_secret` (for server-to-server clients), and a **scope ceiling** that caps what it can ever do.

Access token

You exchange your credentials for a short-lived, opaque **bearer token** and send it on every request. Tokens expire (one hour by default); you request a new one when they do.

Scope

A **scope** is a permission of the form `resource:read` or `resource:write` (e.g. `orders:read`). A token only does what its scopes allow, and never more than the client's ceiling. `read` never implies `write`.

Resource

A **resource** is a kind of record you work with: products (and subproducts), orders (and line items), customers, and read-only reference data. Each lives under `https://<your site domain>/Api/rest/`.

Everything is logged

Every call — success or failure — is recorded, and every change you make is captured with before/after state so an administrator can review or roll it back. Build as though someone is watching, because the audit trail is (see §13).

3. Before you begin

Ask your administrator for the following. You cannot self-serve these — they are issued from the back-office **API → API Clients** screen.

You need	Notes
Base URL	The API root, e.g. `https://<your site domain>/Api`. All paths below are relative to it.
`client_id`	Public identifier for your integration (auto-generated).
`client_secret`	Confidential. Server-to-server clients only; shown once at creation/rotation.
Granted scopes	The exact permissions your client may use (e.g. `products:read orders:read`).
IP allow-list (if any)	If set, requests — including token requests — only succeed from these addresses.

> The `client_secret` is shown **once**. Store it in a secret manager or environment variable, never in source control or a browser. If it is lost or leaked, an administrator rotates it — the old one stops working instantly.

4. Quick start (5 minutes)

This is the shortest path from credentials to data, for a server-to-server client.

1. **Get a token.** Exchange your credentials at the token endpoint:

```
curl -X POST https://<your site domain>/Api/oauth/token \
  -d grant_type=client_credentials \
  -d client_id=YOUR_CLIENT_ID \
  -d client_secret=YOUR_CLIENT_SECRET \
  -d "scope=products:read"
```

2. **Read the token** from the JSON response:

```
{ "access_token": "b5a8c3...e6bae", "token_type": "Bearer", "expires_in": 3600, "scope": "proc
```

3. **Call the API** with the token in the `Authorization` header:

```
curl https://<your site domain>/Api/rest/products?limit=5 \
  -H "Authorization: Bearer b5a8c3...e6bae"
```

4. **Read the result** — a paginated envelope:

```
{ "data": { "items": [ { "id": 42, ... } ], "total": 134, "page": 1, "limit": 5 } }
```

5. **Refresh** by requesting a new token when the old one expires (after `expires_in` seconds).

> That is the whole loop: token → call → refresh. Everything else in this manual is detail on top of these four steps.

5. Authentication in depth

The API uses **OAuth2**. Pick the grant that matches your integration.

5.1 Client credentials (server-to-server)

The default for back-end systems with no end user. The client authenticates as itself with its secret. Request only the scopes you need:

```
curl -X POST https://<your site domain>/Api/oauth/token \
  -d grant_type=client_credentials \
  -d client_id=YOUR_CLIENT_ID -d client_secret=YOUR_CLIENT_SECRET \
  -d "scope=products:read orders:read"
```

Omit `scope` to receive your client's full ceiling. This grant returns no refresh token — simply request another when it expires.

5.2 Authorization code + PKCE (acting for a user)

For apps that act on behalf of a signed-in back-office user. Send the user to the authorize endpoint, then exchange the returned `code` for a token. **Public** clients (no secret, e.g. a single-page or mobile app) **must** use PKCE with the `S256` method — the `plain` method is rejected.

1. Create a PKCE `code_verifier` (random string) and its `code_challenge` (`BASE64URL(SHA256(verifier))`).
2. Redirect the user to `https://<your site domain>/Api/oauth/authorize` with `response_type=code` , your `client_id` , a registered `redirect_uri` , `scope` , a random `state` , `code_challenge` and `code_challenge_method=S256` .
3. The user approves; the browser returns to your `redirect_uri` with `code` and your `state` (verify it matches what you sent).
4. Exchange the code:

   ```
   curl -X POST https://<your site domain>/Api/oauth/token \
     -d grant_type=authorization_code \
     -d code=THE_CODE -d redirect_uri=YOUR_REDIRECT_URI \
     -d client_id=YOUR_CLIENT_ID \
     -d code_verifier=YOUR_PKCE_VERIFIER
   ```

> The `redirect_uri` must **exactly** match one registered for your client, and the authorization `code` is single-use and short-lived (about a minute).

5.3 Refresh tokens

When a response includes a `refresh_token`, use it to get a new access token without involving the user again. Refresh tokens **rotate**: each use returns a new refresh token and invalidates the old one — always store the latest.

```
curl -X POST https://<your site domain>/Api/oauth/token \
  -d grant_type=refresh_token -d refresh_token=YOUR_REFRESH_TOKEN \
  -d client_id=YOUR_CLIENT_ID -d client_secret=YOUR_CLIENT_SECRET
```

You may request a *narrower* scope on refresh, never a wider one.

5.4 Using your token

Send it as a bearer token on every API call:

```
curl https://<your site domain>/Api/rest/orders \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

5.5 Revoking a token

To kill a token immediately (e.g. decommissioning an integration), call revoke with your client credentials — you can only revoke your own tokens:

```
curl -X POST https://<your site domain>/Api/oauth/revoke \
  -d token=THE_TOKEN \
  -d client_id=YOUR_CLIENT_ID -d client_secret=YOUR_CLIENT_SECRET
```

6. Working with resources

All resource endpoints live under `https://<your site domain>/Api/rest/`. The full list of paths, scopes and fields is in the **API Reference**; this is the conceptual tour.

Resource	What it is	Scope prefix
Products & subproducts	Catalog items and their variants, with image and document galleries.	`products`
Orders & line items	Orders and the products on them.	`orders`
Customers	Customer records (credentials and payment data are never returned).	`customers`
Reference	Read-only lookups: countries, states/zones, order statuses, delivery methods, currencies.	`reference`

> Subproducts and order line items reuse their parent's scope — `products:*` covers subproducts, `orders:*` covers line items.

Media (images & documents)

A product carries an `images` array and a `documents` array; subproducts carry `images`. Each entry includes a ready-to-use `url` (CDN when configured). Use `url` to display or download; `file_path` is the raw stored path.

7. Reading data

Response shapes

List endpoints return a paginated envelope; single-item endpoints return one object:

```
// list
{ "data": { "items": [ {...}, {...} ], "total": 134, "page": 1, "limit": 50 } }

// single item
{ "data": { "id": 42, ... } }
```

Pagination & search

Parameter	Meaning
`page`	Page number (default 1).
`limit`	Page size, **max 200** (default 50). Larger values are capped.
`q`	Free-text search, where the resource supports it.

Use `total` to know how many records match, and walk `page` until you have them all. Prefer a sensible `limit` (e.g. 50–100) over many tiny requests.

> Fetching one record? Use `GET .../{id}` rather than searching a list — it is faster and clearer.

8. Writing data

Writes use standard HTTP verbs and JSON bodies.

Creating

Send a JSON body of **writable** fields to the resource's `POST` endpoint. The response is the saved object. Requires the matching `:write` scope.

```
curl -X POST https://<your site domain>/Api/rest/products \
  -H "Authorization: Bearer TOKEN" -H "Content-Type: application/json" \
  -d '{ "name": "New widget", "sku": "SKU-001", "price": 19.99 }'
```

Updating

Send only the fields you want to change to `PUT .../{id}`. Omitted fields are left as they are. Server-managed fields (ids, audit columns, tenant) are ignored if sent.

Deleting

Where a resource supports it, `DELETE .../{id}` performs a **soft delete** (marks the record inactive). Add `?hard=1` to remove it permanently.

> A hard delete cannot be undone through the API. Prefer a soft delete unless you are certain. (An administrator can still roll back a change — see §13.)

9. Scopes & permissions

Scopes are the permission system. A token can only do what its scopes allow, and a client can never be granted (nor a token request exceed) the client's **ceiling**.

- `<resource>:read` — list and read that resource.
- `<resource>:write` — create, update and delete it. **Write does not imply read** — request both if you need to read and change a resource.
- The aliases `read` / `write` expand to every `*:read` / `*:write` in your ceiling.

Scope	Grants
orders:read	Read access to **orders**
orders:write	Write access to **orders**
products:read	Read access to **products**
products:write	Write access to **products**
customers:read	Read access to **customers**
customers:write	Write access to **customers**
reference:read	Read access to **reference**

> Request the narrowest scope set that does the job. A token with fewer scopes limits the damage if it is ever exposed.

10. Rate limits & best practices

Calls are rate-limited per client over a fixed time window. Every response tells you where you stand:

Header	Meaning
`X-RateLimit-Limit`	Requests allowed in the current window.
`X-RateLimit-Remaining`	Requests left in this window.
`X-RateLimit-Reset`	Seconds until the window resets.

Exceed the limit and you get `429 Too Many Requests` with a `Retry-After` header (seconds to wait). The token endpoint is additionally throttled per source IP.

Good citizenship

- **Cache your token** for its full lifetime instead of fetching one per call.
- **Honour** `Retry-After` — back off, do not hammer.
- **Page efficiently** — one list call of 100 beats 100 single-item calls.
- **Watch** `X-RateLimit-Remaining` and slow down before you hit zero.

11. Errors & troubleshooting

Errors return a JSON body and a matching HTTP status:

```
{ "error": "insufficient_scope", "error_description": "Requires scope: orders:write" }
```

error	HTTP	Meaning
`invalid_request`	400	A required parameter is missing or malformed.
`invalid_client`	401	Client authentication failed (unknown client or wrong secret).
`invalid_grant`	400	The grant (code/refresh token) is invalid, expired or revoked.
`unauthorized_client`	400	The client may not use this grant type.
`unsupported_grant_type`	400	The grant_type is not supported.
`invalid_scope`	400	A requested scope is unknown or exceeds the client ceiling.
`invalid_token`	401	The access token is missing, unknown or expired.
`insufficient_scope`	403	The token does not carry the scope the endpoint requires.
`rate_limited`	429	Rate limit exceeded; see the Retry-After header.
`api_disabled`	503	The API is currently switched off by an administrator.
`not_found`	404	No such resource/route.
`method_not_allowed`	405	The HTTP method is not allowed for this path.
`server_error`	500	An unexpected error occurred.

Common situations

Symptom	Likely cause & fix
`401 invalid_token`	Token missing, expired or revoked — request a new one.
`403 insufficient_scope`	Your token lacks the required scope — request it (within your ceiling) or ask your administrator.
`403 access_denied`	Your IP is not on the client's allow-list.
`429 rate_limited`	Slow down; wait `Retry-After` seconds.
`503 api_disabled`	An administrator has switched the API off. Try later.

> Error descriptions are deliberately generic for server errors (`500`) so no internal detail leaks. If you hit a persistent `500` , capture the time and request and contact your administrator — the full detail is in the server-side log.

12. Security & your responsibilities

The platform protects the API with hashed secrets, opaque high-entropy tokens, exact redirect-URI matching, PKCE for public clients, per-client IP allow-lists and full audit logging. Your part:

- **Guard the secret.** Never ship it in a browser, mobile binary, or public repo.
- **Use HTTPS only.** Never send a token or secret over plain HTTP.
- **Request least privilege.** Narrow scopes, short-lived tokens.
- **Rotate on suspicion.** If a secret may have leaked, have it rotated immediately.
- **Validate `state`** on the authorization-code flow to prevent CSRF.

All access is logged. Every request to this API, whether successful or not, is recorded — including the endpoint, the client, the source IP address, the date and time, and the response — for security and audit purposes.

Access to this system has been granted to your client. This access is provided at the discretion of Indition and may be changed, suspended, or revoked at any time, without notice. Your continued use of the API is conditional on your compliance with these terms.

Rate limits apply: the system default rate limit applies. Every response includes the X-RateLimit-Limit, X-RateLimit-Remaining and X-RateLimit-Reset headers; exceeding the limit returns an HTTP 429 response with a Retry-After header indicating how long to wait before retrying. Do not retry faster than that header allows.

Your access is not currently restricted by IP address; restrictions may be applied at any time.

Your access is limited to the scopes granted to your client. Any attempt to use a scope, resource, or operation outside of your grant will be rejected.

You are responsible for keeping your client credentials confidential. Never embed a client secret in a browser, mobile application, or any other untrusted location. If you believe your credentials have been compromised, contact your administrator immediately so they can be rotated.

This API and the data it exposes are provided “as is”, without warranty of any kind. You agree not to use it to store or transmit unlawful content, to attempt to access data outside your authorisation, to interfere with or place undue load on the system, or to reverse engineer or circumvent its security controls. Misuse may result in the immediate and permanent revocation of access.

13. Auditing & change history

Two records are kept for everything your integration does:

- **Request log** — every call (success or failure): the endpoint, your client, source IP, timestamp, status and timing.
- **Change history** — every create/update/delete, with the before and after state, so an administrator can review or **roll back** a change.

> Rollback is a back-office action only — there is no API to undo a change. Treat writes as real and final from the integration's side.

14. Your documentation & support

Your administrator can generate a copy of these guides **tailored to your client** — showing only the resources and scopes you have been granted, with your specific rate limit and IP rules in the terms of access. Ask for your client-specific Auth Guide and API Reference.

When you report an issue, include: the time (with timezone), the endpoint and method, your `client_id` (never your secret), and the `error` code you received. That lets an administrator find your exact request in the log in seconds.

15. FAQ

How long does a token last?

One hour by default (see `expires_in`). Request a new one when it expires; cache it until then.

Do I need a refresh token?

Only for the authorization-code flow. Client-credentials clients just request a new token.

Why am I getting 403 when my token works elsewhere?

Either the endpoint needs a scope your token lacks (`insufficient_scope`) or your IP is not allow-listed (`access_denied`).

Can I set the record id when creating?

No — ids and other server-managed fields are assigned by the platform and ignored if sent.

Is there a sandbox?

Ask your administrator for a non-production base URL and a test client.

16. Glossary

Term	Meaning
Bearer token	The access token you send in the `Authorization` header.
Client	Your integration's registered identity (`client_id` + secret).
Scope	A permission like `orders:read` .
Ceiling	The maximum set of scopes your client may ever use.
PKCE	Proof Key for Code Exchange — secures the auth-code flow for public clients.
Soft / hard delete	Mark inactive vs. remove permanently (`?hard=1`).
Envelope	The `{ "data": { ... } }` wrapper around every response.