ENRICH

360° CUSTOMER INTELLIGENCE

API VERSION 1

Official API Documentation

Programmatic identity, business and KYC verification. Run AI-driven verification workflows, stream results in real time, and reconcile every credit — from any backend, in any language.

Document revision 1.0 · June 2026
Timble Technologies · Enrich Platform

Contents

1 Introduction

The Enrich API lets your systems run the same AI-driven verifications your team performs in the Enrich web application — identity, KYC/KYB, employment, mobile, vehicle and business due-diligence checks — fully programmatically.

The API is **run-based**: you create a *run* (one verification request), Enrich executes it asynchronously through its agent and data providers, and you retrieve the result by ID — by polling, long-polling, or a live event stream. A run's result is durable: it can be fetched at any time after completion, regardless of dropped connections, client crashes or retries.

Base URL

```
https://enrich.timble.ai/api/v1
```

All endpoints in this document are relative to this base. Your account manager provides the host for your environment.

Design guarantees

Guarantee	What it means for you
Durable results	Every answer, report and charge is retrievable by `run_id` forever. Nothing exists only inside a socket.
Idempotent creation	Retrying a create with the same `Idempotency-Key` can never run or bill a verification twice.
Exact billing	Each run reports `credits_used` and a per-tool breakdown that always reconciles with your organisation's balance.
Stable contract	Within `/v1`, changes are additive only — new fields and event types may appear; existing ones are never renamed or removed.

2 Authentication

Every request carries an organisation API key in the `Authorization` header:

```
Authorization: Bearer sk-tm-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Obtaining a key

An organisation administrator creates keys in the Enrich web app under **Admin → API Keys**. The full secret is displayed exactly once at creation — store it in a secret manager. Up to 10 active keys per organisation; each can be named (e.g. *production*, *staging*) and revoked independently and instantly.

Security model

Property	Detail
Scope	Keys work **only** on `/v1/*` endpoints. They are rejected by the web application and admin endpoints, so a leaked key can never manage users, keys or billing.
Identity	Runs created with a key are attributed to your organisation's service identity — not to the administrator who created the key. Keys keep working if that person leaves.
Storage	Enrich stores only a SHA-256 hash of the key; the plaintext cannot be recovered.
Failure mode	Unknown, revoked and expired keys all return the same opaque `401 invalid_api_key`.

3 Quickstart

A complete verification in three calls.

Step 1 — Start the run

```
# the Idempotency-Key makes retries safe
curl -X POST https://enrich.timble.ai/api/v1/runs \
  -H "Authorization: Bearer $ENRICH_KEY" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: order-7841" \
  -d '{
    "message": "Verify identity: Rahul Sharma, +91 98xxxxxx10, PAN ABCDE1234F",
    "workflow": "kyc-verification",
    "metadata": {"order_id": "7841"}
  }'
```

```
HTTP 202
{ "id": "run_9f3a2c…", "thread_id": "th_77b2…", "status": "queued", … }
```

Step 2 — Wait for the result (long-poll)

```
curl "https://enrich.timble.ai/api/v1/runs/run_9f3a2c…?wait=50" -H "Authorization: Bearer $ENRICH_KEY"
```

The call returns as soon as the status changes (or after 50 s). Repeat until the status is terminal.

Step 3 — Read the answer

```
{
  "status": "completed",
  "answer": "PAN verified — name match 96%. Mobile active (Airtel)…",
  "report": { /* structured verification report */ },
  "report_url": "https://…/report.pdf?X-Amz-…",
  "credits_used": 11.8,
  "tool_calls": [
    { "id": "call_201", "name": "pan_verification", "status": "success", "billed": true, "credits": 2.0 }
  ]
}
```

Tip — if the agent proposes a verification plan first, the run pauses at `requires_approval`. See section 7 — approve it with one extra call, or enable auto-approval on the key for zero-touch automation.

4 Core concepts

4.1 Runs & the status lifecycle

A run is one verification request and is the unit of execution, retrieval and billing.

Status	Meaning	Terminal
`queued`	Accepted, about to execute.	—
`running`	The agent is executing checks.	—
`requires_approval`	Paused on a proposed plan; waiting for your decision (24 h timeout).	—
`completed`	Finished; `answer` / `steps` / `report` / `report_url` / `credits_used` populated.	✔
`failed`	Internal error; see `error`. Work performed is billed.	✔
`cancelled`	Cancelled by you, by a declined plan, or by approval timeout.	✔
`interrupted`	Orphaned by a service restart. Safe to retry with the same `Idempotency-Key`.	✔

Allowed transitions: `queued → running → completed | failed`; `running ⇄ requires_approval` (approve resumes, decline cancels); any non-terminal state can become `cancelled` or `interrupted`.

4.2 Threads

Every run belongs to a *thread* — a conversation context. Omit `thread_id` to start a new thread; pass a previous run's `thread_id` to ask follow-up questions with full memory of earlier results (*"Also check his employment history"*). Each run in a thread is billed and retrievable independently.

4.3 Workflows

A workflow is a named, reusable verification procedure the agent follows — which checks to run, in what order, and what the report must emphasise. Pass its slug as `workflow` when creating a run.

- **Built-in workflows** ship with Enrich: `kyc-verification`, `kyb-us`, `kyc-us`, `employment-verification`, `mobile-verification`, `vehicle-rc-verification`, `uae-verification`, `business-due-diligence`, `social-profile-analysis`.
- **Custom workflows** are created by your organisation's admin (**Admin → Workflows**) as Markdown procedures. They are private to your organisation and take precedence over a built-in with the same slug.

Discover what is available to your key with `GET /v1/workflows` (section 5.7).

4.4 Credits & billing

Verifications consume **credits** from your organisation's prepaid pool — the same pool the web application uses. Each data-provider check has a price; model usage is metered per token. After every run, `credits_used` is the exact amount debited, and `tool_calls[]` itemises each check with its individual price and billing status (failed checks are typically not billed).

Overdraft — the balance is checked when a run is created. A run admitted with a positive balance executes to completion even if it crosses zero, and the organisation owes the overrun (the balance goes negative). Top-ups settle the debt first. Monitor `GET /v1/usage` and your run sizes accordingly.

5 Endpoint reference

5.1 Create a run

`POST` `/v1/runs`

HEADERS

Header	Required	Description
`Authorization`	yes	`Bearer sk-tm-…`
`Idempotency-Key`	recommended	Client-chosen string (≤255 chars), unique per logical request. Replaying the same key + body returns the existing run (HTTP 200) instead of executing again. Same key with a *different* body → `409 idempotency_key_reuse`.

BODY

Field	Type	Required	Description
`message`	string	yes	The verification request in plain language, including all identifiers you have (name, phone, PAN, registration numbers…).
`workflow`	string	no	Workflow slug to follow (section 4.3). Unknown slug → `404 workflow_not_found`.
`thread_id`	string	no	Continue an existing conversation (section 4.2).
`stream`	boolean	no	`true` upgrades this response to a Server-Sent-Events stream (section 6). Default `false`.
`metadata`	object	no	Up to 2 KB of your own data, echoed back verbatim on every read (e.g. your order ID).

RESPONSE — 202 ACCEPTED (200 ON IDEMPOTENT REPLAY)

The full run resource (see section 5.2). Execution proceeds in the background.

5.2 Retrieve a run

`GET` `/v1/runs/{run_id}?wait={seconds}`

Parameter	Description
`wait`	Optional long-poll: hold the request up to *N* seconds (max 50) and return early on any status change. Strongly recommended over tight polling loops.

THE RUN RESOURCE

```
{
  "id":           "run_9f3a2c…",
  "thread_id":    "th_77b2…",
  "status":       "completed",
  "input":        { "message": "…", "workflow": "kyc-verification" },
  "answer":       "…",            // final synthesised reply (not the running narration)
  "steps":        [ … ],          // ordered timeline: text + tool_call + report + plan
  "report":       { … },          // structured report, when one was generated
  "report_url":   "https://…",    // presigned report PDF; short-lived, re-minted per read
  "plan":         { … },          // present only while status = requires_approval
  "error":        { "code", "message" },   // present when failed
  "credits_used": 11.8,
  "tool_calls":   [ { "id", "name", "status", "billed", "credits", "latency_ms" } ],
  "metadata":     { … },
  "created_at", "started_at", "completed_at"
}
```

5.3 Stream run events

`GET` `/v1/runs/{run_id}/events`

Server-Sent-Events stream of the run's full event history and live progress. Honours the standard `Last-Event-ID` header (sent automatically by `EventSource` on reconnect) to replay only what you missed. For finished runs, replays everything and closes. Full event vocabulary in section 6.

5.4 Resolve plan approval

`POST` `/v1/runs/{run_id}/approval`

```
{ "decision": "approved" }   // or "declined"
```

Valid only while the run is `requires_approval` — anything else returns `409 not_awaiting_approval` (which also makes concurrent double-approvals safe). `approved` resumes execution; `declined` cancels the run — the agent performs **no further checks**, and only the planning work already done is billed.

5.5 Cancel a run

`POST` `/v1/runs/{run_id}/cancel`

Stops a `queued`, `running` or `requires_approval` run. Work already performed is billed; partial answers are preserved on the run. Terminal runs return `409 not_cancellable`.

5.6 List runs in a thread

`GET` `/v1/threads/{thread_id}/runs?limit=20&offset=0`

The thread's runs, newest first. `limit` ≤ 100.

5.7 List workflows

`GET` `/v1/workflows`

```
[
  { "slug": "kyc-verification", "name": "Kyc Verification", "scope": "built-in", "is_active": true },
  { "slug": "loan-pre-check",   "name": "Loan applicant pre-check",
    "description": "PAN + phone + employment, then report", "scope": "org", "is_active": true }
]
```

5.8 Get credit balance

`GET` `/v1/usage`

```
{ "credits_used": 188.2, "credits_remaining": 311.8, "max_credits": 500.0 }
```

`credits_remaining` may be **negative** (overdraft owed, section 4.4) or `null` (unlimited plan).

6 Streaming (Server-Sent Events)

Two ways to stream: set `"stream": true` on create (the same response becomes the stream), or attach to `GET /v1/runs/{id}/events` at any time. Both emit the identical, persisted event sequence.

Wire format

```
id: 41
event: answer.delta
data: {"text": "PAN verified — name match 96%…"}

: ping                        ← heartbeat comment every ~15 s of silence
```

Every event has a monotonically increasing `id`. Store the last one you processed; on reconnect, send it as `Last-Event-ID` to resume without loss or duplication. Browsers' `EventSource` does this automatically.

Event types

event	data	Notes
`run.created`	`{run_id, thread_id}`	Always first — capture the IDs immediately.
`run.requires_approval`	`{plan}`	Run paused; see section 7. Stream stays open (heartbeats) while parked.
`tool_call`	`{id, name, status}`	`status` : started · success · failed · not_found.
`answer.delta`	`{text}`	Incremental answer text, batched (~0.7 s granularity).
`run.report`	`{report}`	The full structured report, when one is generated.
`run.completed`	`{status, credits_used, report_url}`	Always last; carries the terminal status (completed / failed / cancelled / interrupted) + presigned report PDF.

Contract — new event types may be added over time. Ignore events you don't recognise; never rely on the absence of an event type.

7 Plan approval (human-in-the-loop)

For multi-step verifications, the agent may propose a **plan** before executing checks — what it intends to verify and how. This is a control point for cost and compliance: the run pauses until your side approves.

Lifecycle

1. Run reaches `requires_approval`. The run resource carries `plan` (`{summary, tasks[]}`); streams receive `run.requires_approval` .
2. You call `POST /v1/runs/{id}/approval` with `approved` or `declined` .
3. **Approved** → execution resumes exactly where it paused (any open event stream continues on the same connection). **Declined** → the run is cancelled; the agent is hard-stopped from performing any further checks; only planning work is billed.
4. No decision within **24 hours** → the run auto-cancels.

Zero-touch automation

If you don't want runs pausing, your organisation admin can enable **auto-approve plans** per API key. Plans are then approved instantly in-flight and the run never enters `requires_approval` .

8 Webhooks

Runs are long — instead of polling, register an endpoint and we POST a signed event when a run changes lifecycle state. Endpoints are per-org (manage via `/v1/webhooks` or Admin → Webhooks).

Events

`run.completed`, `run.failed`, `run.cancelled`, `run.interrupted`, `run.requires_approval`. An endpoint with no `event_types` receives all of them.

Payload

The `data` field is the **full run resource** — identical to `GET /v1/runs/{id}` — wrapped in an event envelope. `report_url` is minted fresh per delivery, so it is always valid.

```
POST <your endpoint>
Webhook-Id: evt_3f9a…              # idempotency key — dedupe on this
Webhook-Timestamp: 1718280000      # reject if > 5 min from now
Webhook-Signature: v1,<base64 hmac>

{ "id": "evt_3f9a…", "type": "run.completed", "created_at": "…",
  "data": { /* the full run resource */ } }
```

Verify the signature

```
signed   = `${WebhookId}.${WebhookTimestamp}.` + <raw body>
expected = "v1," + base64( HMAC_SHA256(secret, signed) )   // constant-time compare
```

Delivery

At-least-once. Ack with `2xx` within 10s; failures retry with backoff (`30s → 6h`, 8 tries) then dead-letter. Persistently failing endpoints are auto-disabled (the org admin is emailed). HTTPS only; URLs resolving to private addresses are rejected.

Manage — `POST /v1/webhooks` (returns the `whsec_…` secret once), `GET` list/fetch, `PATCH` update (`is_active:true` re-enables), `DELETE`, and `GET /v1/webhooks/{id}/deliveries`.

9 Errors

Every error has the same machine-readable envelope:

```
{ "error": { "code": "insufficient_credits", "message": "…", "run_id": "run_…" } }
```

HTTP	code	When / what to do
401	`invalid_api_key`	Unknown, revoked or expired key. Rotate via Admin → API Keys.
402	`insufficient_credits`	Balance ≤ 0 at creation. Top up, then retry (same Idempotency-Key is safe).
404	`not_found`	Run/thread doesn't exist in your organisation.
404	`workflow_not_found`	Unknown or deactivated workflow slug. Check `GET /v1/workflows`.
409	`idempotency_key_reuse`	Key reused with a different body. Use a fresh key for new requests.
409	`not_awaiting_approval`	Approval sent to a run not in `requires_approval` (e.g. already resolved).
409	`not_cancellable`	Cancel sent to a run already terminal.
422	`invalid_request`	Validation failure; the message names the field.
429	`rate_limited`	Respect the `Retry-After` header (section 9).
500	`internal_error`	The run is marked `failed`; work performed is billed. Retry with a fresh Idempotency-Key.

10 Rate limits

Limit	Default	On breach
Run creations per key	60 / minute	`429` with `Retry-After: 60`
Concurrently active runs per key	10 (queued + running; paused approvals don't count)	`429` with `Retry-After: 10`
Reads (GET endpoints)	600 / minute per key	`429` with `Retry-After`

Need higher limits? Contact your account manager — limits are configurable per organisation.

11 Versioning & support

The API is path-versioned. Within `/v1`, all changes are **additive**: new endpoints, optional request fields, response fields and event types may appear at any time. Removals and renames only ever happen in a new version path. Build clients that ignore unknown fields and event types.

For integration support, contact your Enrich account team with the `run_id` of any problematic run — every event, check and charge is auditable on our side by that ID.