KNEO AGENT CLIENT

API Reference

Typed Python SDK + adapter toolkit for the Kneo Agent Platform `/v1`.

Single-document reference for every public class, dataclass, exception, and helper that `kneo-client` exposes. Sibling to the kneo_agent SDK reference; same dark-themed layout, same hand-maintained discipline.

VERSION

v0.1.0

PINNED TO KNEO_SERV

v0.4.0 (/v1 only)

PYTHON

≥ 3.12

The user-facing prose ships separately as the Kneo Agent Client User Guide PDF. Source markdown lives under `docs/user/` on the repo. Architecture lives in `docs/dev/architecture.md` and the design handout PDF.

Table of Contents

Read the front matter first; then jump to any class. The sidebar on screen mirrors this layout; in the printed PDF, navigation is page-numbered through the per-page footer.

PLATFORM ADAPTER

`PlatformClient`

- `HealthClient`
- `RunsClient`
- `HumanTasksClient`
- `AuditClient`
- `CredentialsClient`
- `PoliciesClient`

AGENT ADAPTER

`AgentClient`

- `SpecsClient`

REFERENCE

Compatibility matrix

Guides

KNEO AGENT CLIENT

API Reference Manual

Every public class and helper, organized for both screen reading and printing. Adapted from the kneo_agent SDK reference's dark-themed layout; rendered to PDF via the same Puppeteer pipeline.

v0.1.0 Python ≥ 3.12 kneo_serv /v1 · 0.4.x

What's new in v0.1.0

First public release. Everything in this document is new.

Headline pieces:

- **Full `/v1` coverage.** All 25 platform `/v1` endpoints have hand-rolled wrappers backed by `Transport`. The contract test in `tests/contract/test_path_coverage.py` verifies the wrap is exhaustive against the pinned spec.
- **Async-first with a sync facade.** `KneoClient` and every adapter method are `async def`. `SyncTransport` wraps the async transport in a background thread + event loop via `anyio.from_thread.start_blocking_portal()` for callers that cannot run an event loop.
- **Typed throughout.** Strict `mypy` on `kneo_client.{core, platform, agent}`; PEP 561 `py.typed` marker ships in the wheel.
- **Reproducible release.** Sigstore cosign keyless signing on every artifact; CycloneDX SBOM attached to the GitHub release; PyPI Trusted Publishing.

See the 0.1.0 release notes for the long-form summary.

Introduction

`kneo-client` is the shared client layer behind **Kneo Agent Dashboard** (operations) and **Kneo Agent Studio** (development). It owns the operational semantics of talking to a Kneo Agent Platform instance over its `/v1` HTTP API so that downstream products do not each re-invent them.

Three layers, strict dependency direction *generated* → *core* → *adapters*:

- `kneo_client._generated` — *private.* Generated from the pinned `kneo_serv` OpenAPI spec via `openapi-python-client`. External code must not import from this subpackage.
- `kneo_client.core` — handwritten cross-cutting layer: `Transport`, `Profile`, `ApiKeyAuth`, `RetryPolicy`, idempotency, pagination, request IDs, redacted logging, normalized errors.
- `kneo_client.platform` and `kneo_client.agent` — domain-shaped adapters built on `Transport`. `PlatformClient` for the Dashboard, `AgentClient` for the Studio.

Public entry point is a unified `KneoClient` with `.platform` and `.agent` namespaces backed by a single shared `Transport`.

> **Privacy boundary.** Anything under `kneo_client._generated` is private implementation. Its surface can change between any two `kneo-client` releases without notice. If a generated type or helper is useful to external consumers, it gets re-exported through `core`, `platform`, or `agent`; otherwise it stays internal.

For the architectural rationale see `docs/dev/architecture.md` on the repo. For each major design choice see the ADRs under `docs/dev/adrs/`.

KneoClient

The public entry point. Owns one `Transport` and mounts `.platform` and `.agent` namespaces backed by it. Async context manager; closes its transport on exit.

class KneoClient CLASS

Async client for a Kneo Agent Platform instance.

Attributes

ATTRIBUTE	TYPE	DESCRIPTION
`platform`	`PlatformClient`	Operational surface — runs, traces, audit, credentials, policies, health.
`agent`	`AgentClient`	Development surface — spec validate / compile / explain / policy_report / run.

`profile`	`Profile`	The resolved profile this client is bound to.

```
def __init__(profile: Profile, *, retry_policy: RetryPolicy | None = None) → None
```

Build a `KneoClient` bound to `profile`. The transport is constructed eagerly; nothing happens on the network until the first `.platform.*` or `.agent.*` call.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`profile`	`Profile`	—	Resolved profile carrying URL, API key, scheme, and timeout.
`retry_policy`	`RetryPolicy | None`	`None`	Override the default retry policy (3 attempts, exp backoff). See `RetryPolicy`.

```
@classmethod def from_profile(name: str | None = None, **overrides: Any) → KneoClient
```

Resolve a profile via `load_profile()` and build a `KneoClient`. Convenience for the common case of "use the standard config-file / env / kwargs resolution chain".

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`name`	`str | None`	`None`	Profile name to load. Falls back to `$KNEO_PROFILE` then `"default"`.
`**overrides`	`Any`	—	Forwarded to `load_profile`: `config_file`, `url`, `api_key`, `auth_scheme`, `timeout`.

```
async def aclose() → None
```

Close the underlying transport. Idempotent.

```
async def __aenter__() → KneoClient · async def __aexit__(*exc) → None
```

Async context manager. Use `async with KneoClient.from_profile() as client:` to ensure the transport closes on exit.

Example

```
import asyncio
from kneo_client import KneoClient

async def main():
    async with KneoClient.from_profile() as client:
        ready = await client.platform.health.readyz()
        print(f"ok={ready.ok}")

        run = await client.platform.runs.create({"spec_id": "my-spec"})
        terminal = await client.platform.runs.wait_for_completion(run.run_id,
timeout=120)
        print(f"final={terminal.status}")

asyncio.run(main())
```

Transport

The request engine. `Transport` is the only layer that does I/O; every adapter call eventually routes through its `request()` method.

Owns: the `httpx.AsyncClient`, the auth flow, the retry loop, idempotency-key injection, request-ID injection, redacted logging, and the error-to-exception mapping point.

class Transport CLASS

Async HTTP transport for a Kneo Agent Platform instance.

Attributes

ATTRIBUTE	TYPE	DESCRIPTION
`profile`	`Profile`	The profile this transport is bound to (read-only).

```
def __init__(profile: Profile, *, http_client: httpx.AsyncClient | None = None, retry_policy: RetryPolicy | None = None) → None
```

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`profile`	`Profile`	—	Resolved profile.
`http_client`	`httpx.AsyncClient | None`	`None`	Optional pre-built client. When supplied, the caller owns its lifecycle; `aclose()` will not close it. Base URL and auth are *not* overridden on a passed-in client.
`retry_policy`	`RetryPolicy | None`	`None`	Defaults to `RetryPolicy()` (3 attempts, exp backoff with jitter).

```
async def request(method: str, path: str, *, json: Any = None, params: Mapping | None = None, headers: Mapping | None = None, idempotency_key: str | None = None, request_id: str | None = None) → httpx.Response
```

Send an HTTP request with auth, retries, idempotency, and error mapping.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`method`	`str`	—	HTTP method (`GET` , `POST` , …).
`path`	`str`	—	Path relative to the profile's base URL (e.g. `/v1/runs`).

`json`	`Any`	`None`	JSON body.
`params`	`Mapping[str, Any] | None`	`None`	Query string parameters.
`headers`	`Mapping[str, str] | None`	`None`	Extra headers. Auth, `Idempotency-Key`, and `X-Request-ID` are added automatically and override anything in this mapping.
`idempotency_key`	`str | None`	`None`	Override the auto-generated `POST` idempotency key. Validated against the 256-character platform limit. Ignored on non-`POST` methods.
`request_id`	`str | None`	`None`	Override the auto-generated request ID.

RETURNS

`httpx.Response` — the successful response (status < 400).

RAISES

A `KneoError` subclass after retries are exhausted. See Exceptions for the mapping.

Retry behavior

The transport retries on:

- Transport-level errors from `httpx` (DNS, connect, read timeout, TLS).
- HTTP status in `RETRYABLE_STATUS_CODES = {429, 502, 503, 504}`.

Retries fire only for:

- **Idempotent verbs** — `GET`, `HEAD`, `OPTIONS`, `PUT`, `DELETE`.
- **`POST` with an idempotency key** — which is always, since the transport auto-injects a UUID4 key on every `POST`.

A `Retry-After` response header on 429 overrides the computed delay verbatim (jitter is not applied on top).

```
async def aclose() → None
```

Close the underlying `httpx.AsyncClient` if this `Transport` owns it. When a caller-supplied `http_client` was passed to `__init__`, the caller's client is not closed.

```
async def __aenter__() · async def __aexit__(*exc)
```

Async context manager. `async with Transport(profile) as t:` closes the transport on exit.

Example: direct use (no adapter)

```python
import asyncio
from kneo_client.core.profiles import load_profile
from kneo_client.core.transport import Transport

async def main():
    profile = load_profile()
    async with Transport(profile) as t:
        resp = await t.request("GET", "/v1/healthz")
        print(resp.json())

asyncio.run(main())
```

SyncTransport

Synchronous facade around `Transport`. Runs the async transport in a dedicated background thread + event loop via `anyio.from_thread.start_blocking_portal()`. Each call dispatches into that loop and blocks until the coroutine returns.

For callers that cannot run an event loop (scripts, notebooks, sync frameworks). The async surface remains the recommended path.

class SyncTransport CLASS

```
def __init__(profile: Profile, *, retry_policy: RetryPolicy | None = None) →
```

```
None
```

```
def request(method: str, path: str, *, json=None, params=None, headers=None,
idempotency_key=None, request_id=None) → httpx.Response
```

Synchronous version of `Transport.request`. Same arguments, same semantics; blocks until done.

```
def close() → None · def __enter__() · def __exit__(*exc)
```

Lifecycle. Use `with SyncTransport(profile) as t:` to close the background portal on exit.

> **Caveat.** `SyncTransport` does **not** mount `.platform / .agent` adapters. Those are async-only. Sync consumers wanting wrapped endpoints either glue async-to-sync at the call site, or drop to `t.request(method, path, …)` directly.

Example

```
from kneo_client.core.profiles import load_profile
from kneo_client.core.transport import SyncTransport

with SyncTransport(load_profile()) as t:
    resp = t.request("GET", "/v1/healthz")
    print(resp.json())
```

Profile & profiles

A *profile* is a frozen dataclass bundling `(name, url, api_key, auth_scheme, timeout)` — everything `Transport` needs to talk to one Kneo Agent Platform instance.

class Profile DATACLASS

`frozen=True`. Pass directly to `KneoClient(profile)` or build via `load_profile()`.

FIELD	TYPE	DEFAULT	DESCRIPTION
name	str	—	Profile name. Informational; used in

			log records.
`url`	`str`	—	Base URL of the platform instance.
`api_key`	`str`	—	API key to authenticate with. Treat like any other secret; the redaction-aware logger masks it in header output.
`auth_scheme`	`AuthScheme`	`BEARER`	How to present the key (`Authorization: Bearer` vs `X-Kneo-Api-Key`).
`timeout`	`float`	`30.0`	Per-request timeout in seconds.

class ProfileError EXCEPTION

Raised when a profile cannot be resolved (missing `url / api_key` , malformed TOML, bad scheme, non-numeric timeout, etc.).

load_profile() FUNCTION

```
def load_profile(name: str | None = None, *, config_file: Path | None = None,
url: str | None = None, api_key: str | None = None, auth_scheme: AuthScheme |
str | None = None, timeout: float | None = None) → Profile
```

Resolve a `Profile` from TOML config + env vars + explicit kwargs. Resolution order (later overrides earlier):

1. TOML at `config_file` (default: `~/.config/kneo/client.toml` via `platformdirs`).
2. Environment variables: `KNEO_URL` , `KNEO_API_KEY` , `KNEO_AUTH_SCHEME` , `KNEO_TIMEOUT` . `KNEO_PROFILE` selects which TOML section to load.
3. Explicit keyword arguments to this function.

PARAMETER	TYPE	DEFAULT	DESCRIPTION

`name`	`str | None`	`None`	Profile name. Falls back to `$KNEO_PROFILE`, then `"default"`.
`config_file`	`Path | None`	`None`	Override the default TOML location. Non-existent files are skipped silently.
`url`, `api_key`, `auth_scheme`, `timeout`	`str` / `AuthScheme` / `float`	`None`	Override the resolved field. `auth_scheme` accepts either an enum or its string value.

RETURNS

`Profile` — fully resolved.

RAISES

`ProfileError` — if `url` or `api_key` cannot be resolved from any source, or if the config file is malformed.

```
def default_config_path() → Path
```

Return the XDG-style default location of `client.toml` (`~/.config/kneo/client.toml` on Linux, `~/Library/Application Support/kneo/client.toml` on macOS, etc., via `platformdirs.user_config_dir("kneo")`).

Environment-variable constants

CONSTANT	VALUE	PURPOSE
`DEFAULT_PROFILE_NAME`	`"default"`	Profile name when none is specified.
`DEFAULT_TIMEOUT_SECONDS`	`30.0`	Default per-request timeout.
`ENV_PROFILE`	`"KNEO_PROFILE"`	Selects which TOML section to load.
`ENV_URL`	`"KNEO_URL"`	Override profile `url`.
`ENV_API_KEY`	`"KNEO_API_KEY"`	Override profile `api_key`.

`ENV_AUTH_SCHEME`	`"KNEO_AUTH_SCHEME"`	Override profile `auth_scheme`.
`ENV_TIMEOUT`	`"KNEO_TIMEOUT"`	Override profile `timeout`.

Example: TOML config

```toml
# ~/.config/kneo/client.toml
[default]
url = "https://kneo.example.com"
api_key = "prod-key"
auth_scheme = "bearer"
timeout = 30.0

[staging]
url = "https://staging-kneo.example.com"
api_key = "staging-key"
```

Example: resolve a profile

```python
from kneo_client.core.profiles import load_profile

p = load_profile()                    # 'default' from TOML + env
p = load_profile("staging")           # explicit profile
p = load_profile(url="https://ad-hoc", api_key=tok)  # explicit kwargs win
```

ApiKeyAuth & AuthScheme

The platform accepts the API key as either `Authorization: Bearer <key>` or `X-Kneo-Api-Key: <key>`. `ApiKeyAuth` is an `httpx.Auth` subclass that injects the chosen header on every outgoing request; `AuthScheme` is the enum that picks which.

class AuthScheme ENUM

String-valued enum.

MEMBER	VALUE	HEADER INJECTED
BEARER	"bearer"	Authorization: Bearer <key>
KNEO_API_KEY	"kneo_api_key"	X-Kneo-Api-Key: <key>

class ApiKeyAuth CLASS

Inherits from `httpx.Auth`. Compatible with both `httpx.Client` and `httpx.AsyncClient` because httpx's auth flow is a synchronous generator.

```
def __init__(api_key: str, scheme: AuthScheme = AuthScheme.BEARER) → None
```

PARAMETER	TYPE	DEFAULT	DESCRIPTION
api_key	str	—	Non-empty API key. Empty values raise `ValueError`.
scheme	AuthScheme	BEARER	Header scheme.

```
def auth_flow(request: httpx.Request) → Generator[httpx.Request, httpx.Response, None]
```

Injects the API key header and yields the request unchanged. Called by httpx for every redirect / retry attempt at the transport-level layer.

RetryPolicy

A frozen dataclass describing when and how long to wait between attempts. `Transport` applies it; the policy itself does no I/O.

class RetryPolicy DATACLASS

`frozen=True`.

FIELD	TYPE	DEFAULT	DESCRIPTION

`max_attempts`	`int`	`3`	Total attempts including the first try. Set to 1 to disable retries.
`base_delay`	`float`	`0.2`	Seconds to wait before the second attempt. Each subsequent attempt doubles up to `max_delay`.
`max_delay`	`float`	`30.0`	Cap on the computed delay before jitter.
`jitter`	`float`	`0.1`	Fraction of the delay to randomize by, in `[0, 1]`. With `jitter=0.1` and a 1-second delay, the actual sleep is uniformly in `[0.9, 1.1]`.

```
def delay_for(attempt: int, retry_after: float | None = None) → float
```

Compute the sleep duration before `attempt` (1-indexed). `attempt=1` returns 0 (no delay before the first try).

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`attempt`	`int`	—	The upcoming attempt number, starting at 2.
`retry_after`	`float | None`	`None`	Optional server-supplied hint (from a `Retry-After` header). When set, returned verbatim; jitter is not applied (the server's hint is authoritative).

RETRYABLE_STATUS_CODES

Module constant: `frozenset({429, 502, 503, 504})`. Intentionally not configurable; if your platform deployment legitimately returns transient `500`s, fix the platform.

Example

```
from kneo_client import KneoClient
from kneo_client.core.retries import RetryPolicy

# Aggressive policy for a flaky network
client = KneoClient(profile, retry_policy=RetryPolicy(max_attempts=8,
base_delay=0.5))

# No retries (first failure surfaces immediately)
client = KneoClient(profile, retry_policy=RetryPolicy(max_attempts=1))
```

Idempotency helpers

Every `POST` the client sends carries an `Idempotency-Key`. `Transport` auto-injects a fresh UUID4 per request unless the caller supplies one via the method's `idempotency_key=` parameter.

The platform short-circuits a duplicate `POST` with the same key + identical payload, returning the original response. Reuse of the same key with a different payload yields HTTP 409, surfaced as `KneoIdempotencyMismatchError`.

CONSTANT	VALUE	PURPOSE
`IDEMPOTENCY_KEY_HEADER`	`"Idempotency-Key"`	The header name as the platform expects it.
`MAX_KEY_LENGTH`	`256`	Platform-enforced maximum length for the value.

```
def new_idempotency_key() → str
```

Return a fresh UUID4-based idempotency key as a string.

```
def validate_idempotency_key(key: str) → None
```

Validate an externally-supplied key. Raises `ValueError` if empty or longer than `MAX_KEY_LENGTH`.

Pagination

Platform list endpoints accept `limit` (1–1000, default 100), `offset`, `sort_by`, `sort_order`. Responses carry `count`, `total`, `limit`, `offset`, `sort_by`, `sort_order` plus the items array.

`PaginatedResult` is a generic wrapper; `iterate_all()` is an async iterator that walks pages transparently.

class PaginatedResult[T] DATACLASS

FIELD	TYPE	DESCRIPTION
`items`	`list[T]`	Items on this page.
`total`	`int`	Total items across all pages, as reported by the server.
`limit`	`int`	Page size requested.
`offset`	`int`	Offset of the first item on this page.
`sort_by`	`str | None`	Sort field, if specified by the server.
`sort_order`	`str | None`	`"asc"` or `"desc"`.

Properties

PROPERTY	TYPE	DESCRIPTION
`count`	`int`	`len(items)`.
`has_more`	`bool`	`offset + count < total`.

```
async def iterate_all(fetch_page: Callable[[int, int],
Awaitable[PaginatedResult[T]]], *, page_size: int = 100, start_offset: int =
```

```
0) → AsyncIterator[T]
```

Walk all items across pages. `fetch_page(limit, offset)` must return a `PaginatedResult`. Clamps `page_size` to `MAX_PAGE_SIZE = 1000`.

Constants: `DEFAULT_PAGE_SIZE = 100`, `MAX_PAGE_SIZE = 1000`.

> **Note.** The list adapter methods on `PlatformClient` return the raw generated response model today (not a `PaginatedResult`). To use `iterate_all()`, wrap the adapter call with a small adapter that builds a `PaginatedResult` from the response. See the Pagination recipe below.

Request-ID helpers

Every request carries an `X-Request-ID`. The transport auto-injects a UUID4; callers can override via the method's `request_id=` parameter for cross-service correlation. The platform echoes the ID on responses and in audit events.

CONSTANT	VALUE
`REQUEST_ID_HEADER`	`"X-Request-ID"`

```
def generate_request_id() → str
```

Return a fresh UUID4-based request ID as a string.

Logging helpers

Loggers under the `kneo_client.*` namespace use the standard `logging` machinery. All header payloads are passed through `redact_headers()` before any sink sees them.

CONSTANT	VALUE
`REDACTED`	`"<redacted>"`

```
def get_logger(name: str = "kneo_client") → logging.Logger
```

Return a logger under the `kneo_client.*` hierarchy. Bare names like `"transport"` are namespaced to `kneo_client.transport`; already-namespaced names (e.g. `"kneo_client.foo"`) pass through unchanged.

```
def redact_headers(headers: Mapping[str, str]) → dict[str, str]
```

Return a copy of `headers` with sensitive values replaced by `<redacted>`. Sensitive header tokens (case-insensitive, substring match): `authorization`, `x-kneo-api-key`, `cookie`, `set-cookie`, `proxy-authorization`.

> **Body redaction is the caller's responsibility.** The redaction layer masks headers, not bodies. If you log request / response payloads, filter them at your application's logging-formatter layer.

Exceptions

Every failure surfaces as a typed exception derived from `KneoError`. Each one carries the original HTTP status, response body (parsed JSON when possible), the server-assigned request ID, and — for `POST` failures — the idempotency key that was sent.

Hierarchy

```
KneoError ├── KneoNetworkError # transport-level: DNS / connect / TLS / read timeout ├── KneoAuthError # HTTP 401 ├── KneoPermissionError # HTTP 403 ├── KneoNotFoundError # HTTP 404 ├── KneoConflictError # HTTP 409 (generic) │ └── KneoIdempotencyMismatchError # HTTP 409 on idempotency-key replay with different payload ├── KneoRateLimited # HTTP 429 (carries .retry_after) └── KneoServerError # HTTP 5xx
```

class KneoError EXCEPTION

Base exception. Inherits from `Exception`.

ATTRIBUTE	TYPE	DESCRIPTION
`status`	`int | None`	HTTP status, or `None` for transport-level failures.

`body`	`Any`	Parsed JSON dict, raw text, or `None`.
`request_id`	`str | None`	The `X-Request-ID` the platform returned, when present.
`idempotency_key`	`str | None`	The `Idempotency-Key` sent on the failing request, when set.

class KneoRateLimited EXCEPTION

Adds one attribute on top of `KneoError`:

ATTRIBUTE	TYPE	DESCRIPTION
`retry_after`	`float | None`	Seconds parsed from the `Retry-After` header.

from_response() FUNCTION

```
def from_response(response: httpx.Response, *, idempotency_key: str | None = None) → KneoError
```

Map an HTTP response to the appropriate `KneoError` subclass.

HTTP status	Exception	Notes
401	`KneoAuthError`	Missing or invalid API key.
403	`KneoPermissionError`	Key valid; insufficient scope.
404	`KneoNotFoundError`	Resource missing.
409 (with key)	`KneoIdempotencyMismatchError`	Idempotency-key replay with different payload.
409 (no key)	`KneoConflictError`	Generic conflict.
429	`KneoRateLimited`	Carries `retry_after`.
5xx	`KneoServerError`	Server-side failure.

Other	`KneoError`	Catch-all.
Transport	`KneoNetworkError`	Wrapped from `httpx.HTTPError`.

Example

```
from kneo_client.core.errors import KneoError, KneoIdempotencyMismatchError,
KneoRateLimited

try:
    run = await client.platform.runs.create(payload, idempotency_key=key)
except KneoIdempotencyMismatchError as exc:
    # Same key reused with a different payload — almost always a caller bug.
    log.error("mismatch on key=%r body=%r", exc.idempotency_key, exc.body)
    raise
except KneoRateLimited as exc:
    await asyncio.sleep(exc.retry_after or 10)
except KneoError as exc:
    log.error("create_run failed status=%s rid=%s", exc.status, exc.request_id)
    raise
```

PlatformClient

Operational surface for **Kneo Agent Dashboard**. Aggregates six sub-clients backed by a shared `Transport`.

class PlatformClient CLASS

ATTRIBUTE	TYPE	DESCRIPTION
`health`	`HealthClient`	`/v1/{healthz,livez,readyz}`
`runs`	`RunsClient`	`/v1/runs` + 11 sub-endpoints

human_tasks	HumanTasksClient	/v1/human-tasks family
audit	AuditClient	/v1/audit-events
credentials	CredentialsClient	/v1/security/credentials
policies	PoliciesClient	/v1/policies/environment family

```
def __init__(transport: Transport) → None
```

Build the aggregator and attach all six sub-clients. Usually you don't construct one directly — `KneoClient(profile).platform` gives you one.

HealthClient

Wraps the three platform health probes.

class HealthClient CLASS

```
async def healthz() → HealthResponse
```

GET `/v1/healthz` — overall service health.

```
async def livez() → HealthResponse
```

GET `/v1/livez` — process liveness. Does *not* check downstream deps.

```
async def readyz() → HealthResponse
```

GET `/v1/readyz` — readiness (database, queue, runtime registry, providers, MCP).

Response (HealthResponse)

FIELD	TYPE	DESCRIPTION
ok	bool	Overall pass/fail.

`service`	`str | UNSET`	Service name, e.g. `"kneo-serv-platform"`.
`version`	`str | UNSET`	Platform version.
`metadata`	`HealthResponseMetadata | UNSET`	Per-check breakdown.

RunsClient

The largest sub-client — 12 endpoints + one convenience helper (`wait_for_completion`). Covers the full run lifecycle: create, list, get, cancel, continue, replay, trace, checkpoints, policy reports, recovery.

class RunsClient CLASS

```
async def create(body: RunCreateRequest | dict, *, idempotency_key: str | None = None) → RunCreateResponse
```

POST `/v1/runs` — start a new run. Idempotency key is auto-generated unless caller supplies one.

```
async def list(*, status: str | None = None, limit: int = 100, offset: int = 0, sort_by: str = "updated_at", sort_order: str = "desc") → RunListResponse
```

GET `/v1/runs` — list runs with limit/offset pagination.

```
async def get(run_id: str) → RunStatusResponse
```

GET `/v1/runs/{run_id}` — fetch the current status of a run.

```
async def cancel(run_id: str, *, idempotency_key: str | None = None) → RunStatusResponse
```

POST `/v1/runs/{run_id}/cancel` — cooperatively cancel a running run. The platform may not honor the cancel if the run is already in a terminal state; the returned status reflects the post-cancel state.

```
async def continue_(run_id: str, *, idempotency_key: str | None = None) → RunCreateResponse
```

POST `/v1/runs/{run_id}/continue` — resume a paused run (typically after a human task was resumed). Trailing underscore avoids the Python `continue` keyword collision.

```
async def replay(run_id: str) → RunReplayResponse
```

GET `/v1/runs/{run_id}/replay` — fetch a deterministic replay view derived from the run's checkpoints + trace.

```
async def trace(run_id: str, *, event_type: str | None = None, limit: int = 100, offset: int = 0, sort_by: str = "timestamp", sort_order: str = "asc") → TraceResponse
```

GET `/v1/runs/{run_id}/trace` — fetch the event trace for a run (tool calls, model calls, middleware decisions, etc.).

```
async def checkpoints(run_id: str, *, limit: int = 100, offset: int = 0, sort_by: str = "sequence", sort_order: str = "asc") → CheckpointListResponse
```

GET `/v1/runs/{run_id}/checkpoints` — list a run's serialized state snapshots.

```
async def checkpoints_diff(run_id: str, *, from_sequence: int | None = None, to_sequence: int | None = None) → CheckpointDiffResponse
```

GET `/v1/runs/{run_id}/checkpoints/diff` — diff two checkpoints.

```
async def policy_report(run_id: str) → dict[str, Any]
```

GET `/v1/runs/{run_id}/policy-report` — fetch policy outcomes for a run. Returns the raw JSON body; the platform's policy-report schema is intentionally open-ended and not modeled as a typed response.

```
async def recovery(run_id: str) → RunRecoveryResponse
```

GET `/v1/runs/{run_id}/recovery` — fetch the recovery context for a failed run (what state can be recovered, what next action is recommended).

```
async def wait_for_completion(run_id: str, *, timeout: float | None = None, poll_interval: float = 1.0, terminal_statuses: Iterable[str] | None = None) → RunStatusResponse
```

Poll `get(run_id)` until the run reaches a terminal status. Convenience helper; not a separate platform endpoint.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`timeout`	`float | None`	`None`	Total time budget in seconds. `None` means wait indefinitely.
`poll_interval`	`float`	`1.0`	Seconds to sleep between polls.
`terminal_statuses`	`Iterable[str] | None`	`None`	Defaults to `{"completed", "failed", "cancelled"}`. Pass `{"paused_human_review", ...}` to treat additional states as terminal.

RETURNS

The terminal `RunStatusResponse`.

RAISES

`TimeoutError` if `timeout` elapses before a terminal status is reached.

Module constant: `DEFAULT_TERMINAL_STATUSES = frozenset({"completed", "failed", "cancelled"})`.

Example: end-to-end run

```python
async with KneoClient.from_profile() as client:
    created = await client.platform.runs.create({"spec_id": "my-spec"})
    terminal = await client.platform.runs.wait_for_completion(
        created.run_id, poll_interval=2.0, timeout=600
    )
    print(f"final={terminal.status}")

    trace = await client.platform.runs.trace(created.run_id, limit=20)
    for ev in trace.events:
        print(ev)
```

HumanTasksClient

Human-in-the-loop pause points. A run that requires operator review surfaces as a *human task*, identified by a `continuation_id`. Resuming posts a decision and unblocks the paused continuation.

class HumanTasksClient CLASS

```python
async def list(*, status: str | None = None, limit: int = 100, offset: int = 0, sort_by: str = "created_at", sort_order: str = "desc") → HumanTaskListResponse
```

GET `/v1/human-tasks` — list pending and recent human tasks.

```python
async def get(continuation_id: str) → HumanTaskResponse
```

GET `/v1/human-tasks/{continuation_id}` — fetch a single human task.

```python
async def resume(continuation_id: str, body: HumanResumeRequest | dict, *, idempotency_key: str | None = None) → HumanResumeResponse
```

POST `/v1/human-tasks/{continuation_id}/resume` — resume a paused run with a decision.

AuditClient

Audit events are the canonical operational log: every run, human-task, credential, and policy mutation produces one.

class AuditClient CLASS

```
async def list(*, event_type: str | None = None, run_id: str | None = None, principal: str | None = None, limit: int = 100, offset: int = 0, sort_by: str = "timestamp", sort_order: str = "desc") → AuditEventListResponse
```

GET `/v1/audit-events` — list audit events with three optional filters (event type, run, principal).

CredentialsClient

Lists the credential *references* the platform knows about. The platform never returns raw secret material via the HTTP API.

class CredentialsClient CLASS

```
async def list() → CredentialInventoryResponse
```

GET `/v1/security/credentials` — list known credential references.

PoliciesClient

Environment policies map deployment targets (e.g. `dev`, `staging`, `prod`) to policy bundles that gate which specs / agents are allowed to run there.

class PoliciesClient CLASS

```
async def environment_list() → EnvironmentPolicyListResponse
```

GET `/v1/policies/environment` — list policies for every environment.

```
async def environment_get(environment: str) → EnvironmentPolicyResponse
```

GET `/v1/policies/environment/{environment}` — fetch one environment's policy.

```
async def environment_put(environment: str, body: EnvironmentPolicyRequest | dict) → EnvironmentPolicyResponse
```

PUT `/v1/policies/environment/{environment}` — replace an environment's policy.

> **Note.** `PUT` is idempotent by HTTP semantics; the transport does not inject an `Idempotency-Key` for it.

AgentClient

Development surface for **Kneo Agent Studio**. Currently aggregates one sub-client (`SpecsClient`).

class AgentClient CLASS

ATTRIBUTE	TYPE	DESCRIPTION
specs	SpecsClient	/v1/specs/*

SpecsClient

Spec validation, compilation, explanation, policy-report, and ad-hoc dry-run. The Studio iterate-and-test loop.

class SpecsClient CLASS

```
async def validate(body: SpecValidateRequest | dict, *, idempotency_key: str | None = None) → SpecValidateResponse
```

POST `/v1/specs/validate` — schema + semantic validation of a spec.

```
async def compile(body: SpecCompileRequest | dict, *, idempotency_key: str | None = None) → SpecCompileResponse
```

POST `/v1/specs/compile` — compile a spec to its runtime representation.

```
async def explain(body: SpecExplainRequest | dict, *, idempotency_key: str | None = None) → SpecExplainResponse
```

POST `/v1/specs/explain` — human-readable summary of a spec.

```
async def policy_report(body: SpecPolicyReportRequest | dict, *, idempotency_key: str | None = None) → SpecPolicyReportResponse
```

POST `/v1/specs/policy-report` — preview policy outcomes for a spec.

```
async def run(body: RunCreateRequest | dict, *, idempotency_key: str | None = None) → RunCreateResponse
```

POST `/v1/specs/run` — ad-hoc dry-run of a spec. Distinct from `RunsClient.create`: `specs.run` accepts an inline spec rather than a spec reference, and is intended for Studio's iterate-and-test flow.

Example: validate-then-compile

```python
async with KneoClient.from_profile() as client:
    payload = {"spec": spec_text}
    validated = await client.agent.specs.validate(payload)
    if not validated.valid:
        for d in validated.diagnostics or []:
            print(d)
        return
    compiled = await client.agent.specs.compile(payload)
    if not compiled.ok:
        for d in compiled.diagnostics or []:
            print(d)
```

Compatibility matrix

Which `kneo-client` release supports which `kneo_serv` platform version. The platform's `/v1` HTTP API is a stability boundary.

kneo-client	Pinned to kneo_serv	Tested against	Python	Status
0.1.0	`v0.4.0` (`info.version` `0.4.0`)	`kneo_serv` `0.4.x` line	≥ 3.12	Current

Forward compatibility

A newer `kneo_serv` that adds endpoints will still work for everything `kneo-client` already wraps. New endpoints are available via the drop-to-transport escape hatch until the next `kneo-client` release wraps them:

```
async with KneoClient.from_profile() as client:
    resp = await client._transport.request("GET", "/v1/some/new/endpoint")
    payload = resp.json()
```

Backward compatibility

`kneo-client X.Y.Z` is **not** guaranteed against `kneo_serv` releases older than its pin. Wrappers may rely on response fields that older versions don't emit, and error mapping assumes the current platform error shape. Pin to a matching `kneo-client` minor when talking to an older platform.

Guides

Wrapping `iterate_all()` around a list endpoint

The platform list methods return the raw generated response model today. To use `iterate_all()`, build a small adapter:

```python
from kneo_client.core.pagination import PaginatedResult, iterate_all

async def all_audit_events(client, **filters):
    async def fetch_page(limit, offset):
        resp = await client.platform.audit.list(limit=limit, offset=offset,
**filters)
        return PaginatedResult(
            items=resp.events,
            total=getattr(resp, "total", 0) or 0,
            limit=limit, offset=offset,
        )
    async for event in iterate_all(fetch_page, page_size=200):
        yield event
```

Bring your own httpx client

For shared connection pools, custom transports, proxy configuration:

```python
import httpx
from kneo_client.core.auth import ApiKeyAuth, AuthScheme
from kneo_client.core.profiles import Profile
from kneo_client.core.transport import Transport

profile = Profile(name="prod", url="https://kneo", api_key="…",
                  auth_scheme=AuthScheme.BEARER, timeout=30.0)
shared = httpx.AsyncClient(
    base_url=profile.url,
    auth=ApiKeyAuth(profile.api_key, profile.auth_scheme),
    timeout=profile.timeout,
    limits=httpx.Limits(max_keepalive_connections=20, max_connections=100),
)
try:
    async with Transport(profile, http_client=shared) as t:
        # t.aclose() will NOT close `shared`.
        ...
finally:
    await shared.aclose()
```

Wait for a non-default terminal status

`wait_for_completion()` treats `{"completed", "failed", "cancelled"}` as terminal by default. To return as soon as the run pauses for human review:

```python
terminal = await client.platform.runs.wait_for_completion(
    run_id,
    poll_interval=2.0,
    timeout=300,
    terminal_statuses={"completed", "failed", "cancelled", "paused_human_review"},
)
```

More recipes

See `docs/dev/extending.md` on the repo for the full 10-recipe set: custom retry policy, custom auth scheme, wrapping a not-yet-adapter-covered endpoint, adding a new wrapped endpoint, custom logging,

profile from a secrets manager, paginated iteration, custom terminal statuses, sync facade, and what NOT to extend.

Kneo Agent Client — API Reference Manual · v0.1.0 · 2026-05-25 · MIT license

Adapted layout and style from the kneo_agent SDK API reference.