ProjectCore

Developer Integration Guide

v1.0 | March 2026 | Confidential

For: Client Engineering Teams

PURPOSE

Everything you need to authenticate, integrate, validate, and go live on ProjectCore. Written for engineers. No setup assumed beyond an API key.

AUDIENCE

Senior engineers and technical leads building production surfaces on ProjectCore.

RELATED DOCUMENTS

- Technical, Security and Trust Overview
- Agents as Co-Workers Doctrine
- ProjectLedger Integration Guide

SECTION 01

What Is ProjectCore?

ProjectCore is the API-first control plane for human-and-agent production systems. It is not a model provider, not a chatbot wrapper, and not a prompt tool. It is the runtime, governance, memory, evaluation, and ledger layer that makes AI systems accountable, correctable, and durable.

The integration surface is five primitives:

Primitive	Method	What It Does
run()	`POST /api/v1/agents/run/simple`	Execute one agent with governed context
promote()	`POST /api/v1/ledger/promote`	Commit a decision to the immutable Ledger
evaluate()	`POST /api/v1/evaluations`	Open a bounded decision window
delegate()	`POST /api/v1/authority/delegations`	Grant agent authority with explicit scope
halt()	`POST /api/v1/agents/:id/halt`	Stop agent execution. Every halt recorded.

No database changes. No schema migrations. No infrastructure changes. ProjectCore sits alongside your application as a sidecar API.

Architecture

```
Your Application
  |
  v
ProjectCore API          (Vercel middle tier)
  |
  v
Firestore                (per-tenant isolated database)
  |
  v
Google Cloud KMS         (asymmetric signing for Artifact 5)
  |
  v
Artifact 5               (sealed, tamper-evident decision bundle)
```

> NOTE: Your data stays in your systems. ProjectCore stores only the decision metadata you explicitly send via promote(). For sensitive data, use opaque references, not raw values.

SECTION 02

Security and Tenant Isolation

ProjectCore enforces tenant-scoped access, request validation, and decision integrity at the API boundary.

Tenant Scope

Tenant and project scope are derived server-side from authentication context and never trusted from client input. Each project operates within an isolated data boundary. Client data is never co-mingled across tenants.

API Key Security

- Keys use a bearer token or managed API key issued during provisioning
- Only SHA-256 hash stored at rest. Plaintext returned once at provisioning.
- Keys scoped to a single organization
- Keys revocable via status flag
- Cross-organization read/write attempts denied with HTTP 403
- No public ingestion endpoints

Authentication Flow

Tenant scope is derived server-side and never trusted from client input. When your API key is presented, the server resolves your organization and routes all operations to your isolated database.

> WARNING: Never share your API key. If compromised, contact your account team immediately for key revocation and reissue. Existing sealed records are not affected by key rotation.

PII Protection

PII is rejected at ingest. The /promote endpoint scans all payloads and hard-rejects entries containing SSNs, credit card numbers, phone numbers, email addresses, and street addresses before they are written to the ledger. This is a fail-early guarantee. Malformed entries do not reach the ledger.

Production Hardening

- HSTS with 2-year max-age
- Content Security Policy restrictions
- X-Frame-Options: DENY
- X-Content-Type-Options: nosniff
- HTTPS-only. All traffic encrypted in transit.
- KMS asymmetric signing is fail-closed. Artifact generation fails if signing fails.

- Unsigned artifacts are not permitted in production.

SECTION 03

Getting Started

Before your integration can work, two things need to exist on the ProjectCore side. Confirm both before writing any code.

Credential Provisioning

Your account team provisions credentials in this order:

Step	What Happens
1	An organization record is created in the system database
2	A sk_p0_ prefixed API key is generated and registered in the Firestore apiKeys collection with your customerId
3	A customer-{orgId} isolated database is created for your tenant
4	Your credentials are sent: API_ENDPOINT, API_KEY, and PROJECT_ID

> NOTE: If your health check fails, the most common cause is a key that was not registered in the Firestore apiKeys collection. Confirm your sk_p0_ key is active with your account team before debugging your application code.

Configure Credentials

Create a .env file in your project root:

```
# .env
P0_API_ENDPOINT=https://your-core-origin.com
P0_API_KEY=sk_p0_your_key_here
P0_PROJECT_ID=your-project-id
```

> WARNING: Never commit your .env file to source control. Each client receives a unique API key and isolated project namespace.

Verify Connectivity

Run a health check before writing any integration code:

```
import requests, os
from dotenv import load_dotenv
```

```
load_dotenv()

headers = {
    "Authorization": f"Bearer {os.getenv('P0_API_KEY')}",
    "Content-Type": "application/json"
}

r = requests.get(
    f"{os.getenv('P0_API_ENDPOINT')}/api/health",
    headers=headers
)
print(r.json())  # Expected: {"status": "ok", "service": "p0-core"}
```

> NOTE: If health check fails: verify your .env is in the same directory where you run your application, and confirm your bearer token or API key is active with your account team.

SECTION 04

The Five Primitives

The complete integration is built on five primitives. The first integration path uses four of them in sequence. Every production system that requires full governance uses all five.

1. evaluate() — Open a Decision Window

Every decision session begins by opening a bounded evaluation window. Nothing can be recorded until this is called. Returns an evaluationId that all subsequent promote() calls require.

```
import requests, os, json

def create_evaluation(name):
    r = requests.post(
        f"{os.getenv('P0_API_ENDPOINT')}/api/v1/evaluations",
        headers=headers,
        json={"name": name}
    )
    return r.json()["evaluationId"]

eval_id = create_evaluation("Loan Approval - APP-12345")
```

2. run() — Execute an Agent

Run one agent for one turn with governed context. Returns a reply, a truth posture classification, and optional trace metadata. Only decisions that are subsequently promoted enter the permanent record.

```
def run_agent(agent_id, message, memory_scope="working"):
    r = requests.post(
        f"{os.getenv('P0_API_ENDPOINT')}/api/v1/agents/run/simple",
```

```
        headers=headers,
        json={
            "agentId": agent_id,
            "message": message,
            "memoryScope": memory_scope  # working or core
        }
    )
    return r.json()

result = run_agent("underwriting-agent-v2", "Evaluate loan APP-12345")
print(result["reply"])
print(result["truthPosture"])  # Known | Inferred | Unknown
```

> NOTE: No promote, no trace. That distinction is intentional. Run produces output. Promote creates the permanent record. Only what is promoted is attributable.

3. promote() — Commit a Decision

The line between thinking and deciding. Writes an immutable, attributed entry to the Ledger inside the open evaluation. Call once for a single decision or N times for multi-agent pipelines.

```
def promote(eval_id, payload):
    r = requests.post(
        f"{os.getenv('P0_API_ENDPOINT')}/api/v1/ledger/promote",
        headers=headers,
        json={"evaluationId": eval_id, **payload}
    )
    return r.json()

promote(eval_id, {
    "type": "decision",
    "authorityMode": "human_led",
    "actor": {"type": "human", "id": "UW-5521"},
    "summary": "Loan approved at 6.20% fixed.",
    "title": "Loan Approval - Greenfield Manufacturing",
    "decisionState": "accepted",
    "tags": ["underwriting", "commercial-loan"],
    "createdBy": "Patricia Yamamoto, Senior Underwriter",
    "idempotencyKey": "txn-APP12345-001"
})
```

4. close_evaluation() — Seal the Window

Permanently closes the evaluation. No further promote() calls accepted. Returns HTTP 409 on any subsequent promote attempt. Irreversible.

```
def close_evaluation(eval_id):
    r = requests.post(
        f"{os.getenv('P0_API_ENDPOINT')}/api/v1/evaluations/{eval_id}/close",
        headers=headers
    )
    return r.json()
```

```
close_evaluation(eval_id)
```

> WARNING: close_evaluation is irreversible. If you need to record additional decisions, open a new evaluation.

5. generate_artifact() — Produce the Sealed Record

Produces Artifact 5: a KMS-signed, tamper-evident JSON bundle containing every decision recorded during the evaluation window. Fails in production if KMS signing fails.

```
def generate_artifact(eval_id):
    r = requests.post(
        f"{os.getenv('P0_API_ENDPOINT')}/api/v1/artifacts/generate",
        headers=headers,
        json={"evaluationId": eval_id}
    )
    return r.json()

artifact = generate_artifact(eval_id)
print(json.dumps(artifact, indent=2))
```

SECTION 05

Complete Integration Example

End-to-end lifecycle from open to sealed artifact. This is the pattern every integration follows.

```
import requests, os
# The examples below use raw HTTP for portability.
# A minimal TypeScript helper exists in the ProjectCore repo.

# 1. Open the decision window
eval_id = create_evaluation("Loan Approval - APP-12345")

# 2. Run the agent
result = run_agent(
    agent_id="underwriting-agent-v2",
    message="Evaluate loan APP-12345 for approval",
    memory_scope="working"
)

# 3. Commit the decision
promote(eval_id, {
    "type": "decision",
    "authorityMode": "human_in_the_loop",
    "actor": {"type": "human", "id": "UW-5521"},
    "summary": "Reviewed agent recommendation. Loan approved at 6.20% fixed.",
    "title": "Loan Approval - Greenfield Manufacturing",
    "decisionState": "accepted",
    "tags": ["underwriting", "commercial-loan"],
    "createdBy": "Patricia Yamamoto, Senior Underwriter",
```

```
    "idempotencyKey": "txn-APP12345-001"
})

# 4. Seal the window
close_evaluation(eval_id)

# 5. Generate the signed forensic bundle
artifact = generate_artifact(eval_id)
```

> Five calls. One sealed artifact. A complete, independently verifiable decision record from a single integration session.

SECTION 06

Payload Reference

Required Fields

Field	Type	Description
type	"decision" or "note"	Binding commitment or informational entry
authorityMode	See Authority Modes below	Who or what has decision authority
actor	{"type": "...", "id": "..."}	The decision-maker
summary	Free text	What was decided and why

Optional Fields

Field	Type	Description
title	String	Short label for the entry
decisionState	"accepted" / "rejected" / "deferred"	Outcome of the decision
tags	Array of strings	Searchable labels
createdBy	String	Human-readable name of decision-maker
truthPosture	"known" / "inferred" / "unknown"	Agent confidence classification
clientTimestamp	ISO 8601 UTC	When the decision was made on your side
Idempotency-Key	HTTP header	Safe retry header. Prevents duplicate entries on retry. Pass as HTTP header, not request body.
opaqueRefs	Array	Evidence references without storing raw data

NOTE: The idempotencyKey field ensures duplicate requests return the original entry without creating duplicates. Always use it in production to handle retries safely.

Authority Modes

Value	Meaning	Example
human_led	A human made the decision. AI may have assisted.	Physician approves treatment
human_in_the_loop	AI agents proposed. A human reviewed and authorized.	6 agents triage, adjuster confirms
agent_autonomous	An AI agent decided with no human in the chain.	Fraud detection auto-suspends

Actor Types

Value	Meaning	Example
human	A named person	{"type": "human", "id": "MD-12847"}
agent	An AI/ML model or agent	{"type": "agent", "id": "fraud-ml-v7"}
service	A deterministic system	{"type": "service", "id": "age-gate-v3"}

Truth Posture

Every agent run returns a truthPosture classification. This governs what can be promoted autonomously.

Value	Meaning	Autonomous Promote?
known	Agent output is grounded in declared evidence	Permitted
inferred	Agent output is reasoned but not directly evidenced	Permitted
unknown	Agent cannot classify its own output confidence	REJECTED

WARNING: agent_autonomous + truthPosture=unknown is always rejected at the promote() boundary. This is a hard governance gate, not a configuration option.

SECTION 07

Multi-Agent Pattern

For AI pipelines with multiple agents, call promote() once per agent before the human decision. Each call creates a separate ledger entry with its own hash. An auditor can trace exactly which agent found what, and who ultimately committed the decision.

```
# Each agent records its finding as a note
for agent_finding in agent_results:
    promote(eval_id, {
        "type": "note",
        "authorityMode": "agent_autonomous",
        "actor": {"type": "agent", "id": agent_finding["agent_id"]},
        "summary": agent_finding["summary"],
        "truthPosture": agent_finding["truth_posture"]
    })

# Then the human records the binding decision
promote(eval_id, {
    "type": "decision",
    "authorityMode": "human_in_the_loop",
    "actor": {"type": "human", "id": "UW-5521"},
    "summary": "Reviewed 6-agent consensus. Approved.",
    "idempotencyKey": "review-APP12345-final"
})
```

The resulting artifact contains the full chain: every agent note, then the human commitment. The authority mode on the final decision entry is human_in_the_loop, which proves human oversight occurred.

> NOTE: Use type: "note" for agent findings and type: "decision" for the binding human commitment. Only "decision" entries count toward your authority map in Artifact 5.

Delegating Autonomous Authority

For fully autonomous pipelines, delegate authority explicitly before the agent runs:

```
# Grant the agent authority to act autonomously within scope
delegate({
    "agentId": "fraud-detection-v3",
    "actorId": "risk-ops-lead-001",
    "authorityMode": "agent_autonomous",
    "scope": "fraud_suspension",
    "expiresAt": "2026-12-31T23:59:59Z"
})

# Agent-autonomous promote requires an active delegation record and a
non-unknown truth posture
promote(eval_id, {
    "type": "decision",
    "authorityMode": "agent_autonomous",
```

```
    "actor": {"type": "agent", "id": "fraud-detection-v3"},
    "summary": "Account suspended: fraud pattern confidence 98.2%",
    "truthPosture": "known",
    "idempotencyKey": "fraud-acct-88821-001"
})
```

> NOTE: Autonomous promote is rejected if no active delegation exists for the actor/agent pair. Delegation creation is an admin-level operation in current ProjectCore. Contact your account team to configure delegations during onboarding.

SECTION 08

Artifact 5: The Sealed Record

Artifact 5 is the KMS-signed, tamper-evident JSON bundle produced after an evaluation is closed. It is the forensic guarantee. The sealed receipt you hand to any auditor, regulator, or legal team.

What It Contains

- All ledger entries (decisions and notes) with individual SHA-256 hashes
- Policy snapshot at time of sealing
- Evaluation metadata (name, status, timestamps)
- SHA-256 manifestRootHash computed over all entries
- bundleHash over the full sealed bundle
- signature.scheme and signature.keyId
- KMS asymmetric signature metadata where KMS-backed signing is configured

Structure

```
{
  "evaluation": {
    "evaluationId": "eval_9f2c3a8b",
    "name": "Loan Approval - APP-12345",
    "status": "closed",
    "createdAt": "2026-03-01T10:00:00Z",
    "closedAt": "2026-03-01T10:14:22Z"
  },
  "entries": [ ... ],
  "manifestRootHash": "2a9e9d8e4f5c...",
  "bundleHash": "7f3c1b9d...",
  "signature": {
    "scheme": "rsa_sha256",
    "keyId": "projects/.../cryptoKeyVersions/1",
    "value": "MEQCIG8uXJ...",
    "algorithm": "RSA_SIGN_PKCS1_2048_SHA256",
    "signedAt": "2026-03-01T10:14:22Z"
  }
}
```

Independent Verification

Anyone can verify an Artifact 5 bundle without access to ProjectCore infrastructure. You need only the public key.

Step	Action
1	Take the entries array from the bundle
2	Apply documented canonicalization rules to the entries array
3	Recompute the SHA-256 manifest hash from the canonicalized input
4	Compare recomputed hash to manifestRootHash in the artifact
5	Verify the attached signature per the signature.scheme field

```
const recomputed = computeManifestHash(artifact.entries);
const valid = verifySignature(
    artifact.manifestRootHash,
    artifact.signature,
    publicKeyPem
);
// Hash mismatch or signature invalid: tampering detected
// Both match: AUTHENTIC
```

> NOTE: In deployments where KMS-backed signing is configured, Artifact 5 includes asymmetric signature metadata suitable for independent verification against the corresponding public key. Artifact generation is deterministic: the same evaluation always produces the same integrity inputs.

SECTION 09

API Reference

Base URL and Headers

```
POST https://your-core-origin.com/api/{route}
Authorization: Bearer <P0_API_KEY>
Content-Type: application/json
```

Quick Reference Table

Primitive	Route	Purpose

run()	`POST /api/v1/agents/run/simple`	Run one agent for one turn
promote()	`POST /api/v1/ledger/promote`	Commit a decision to the Ledger
create_evaluation()	`POST /api/v1/evaluations`	Open a bounded decision window
close_evaluation()	`POST /api/v1/evaluations/:id/close`	Close and seal the window
generate_artifact()	`POST /api/v1/artifacts/generate`	Seal into Artifact 5
delegate()	`POST /api/v1/authority/delegations`	Create an authority delegation
halt()	`POST /api/v1/agents/:id/halt`	Halt future runs for an agent
suspend()	`POST /api/v1/agents/:id/suspend`	Suspend future runs
resume()	`POST /api/v1/agents/:id/resume`	Re-enable runs
evaluate_agent()	`POST /api/v1/agent-evaluations`	Create an agent evaluation

Response Shape

Success

```
{ "ok": true }
```

Or with operation-specific fields. Most endpoints return top-level fields rather than a consistent data wrapper:

```
{ "ok": true, "evaluationId": "eval_9f2c3a8b", ... }
```

Error

```
{ "ok": false, "error": "Human-readable message", "code": "MACHINE_CODE",
"requestId": "req_..." }
```

Error Codes

HTTP	Code	What To Do
400	BAD_REQUEST	Check required fields in your payload
401	UNAUTHORIZED	Verify your P0_API_KEY is active
403	FORBIDDEN	Cross-tenant access denied. You are accessing another client's data.
409	CONFLICT	Promote after close. Evaluation is sealed. Open a new one.

429	RATE_LIMITED	Back off and retry with exponential backoff
500	INTERNAL_ERROR	Contact your account team

SECTION 10

Data Handling Guidance

ProjectCore is designed to record decision metadata and evidence references rather than raw sensitive application data. It does not execute untrusted code. Keep sensitive application data in your own systems and reference it via opaque identifiers.

DO	DON'T
• Use opaque references (UUIDs, hashed IDs) in summary and tags fields • Store raw sensitive data in your own systems • Reference your internal records via opaque IDs in the promote() payload • Use idempotencyKey on every production promote call	• Include SSNs, credit card numbers, or raw PII in the summary field • Store passwords, API keys, or credentials in ledger entries • Include file paths or internal system URLs • Commit your .env file or API keys to source control

> NOTE: The API scans for common PII patterns at ingest and will hard-reject entries containing emails, phone numbers, credit card patterns, SSNs, or file paths. This rejection happens before anything is written to the ledger.

SECTION 11

Validation Checklist

Run through this checklist after receiving your credentials. Steps 1 through 7 must pass before you are considered live. Steps 8 through 10 apply to production deployments only.

#	Validation Step	Expected Result	Status
1	GET /api/health returns status: ok	`200 OK`	
2	create_evaluation returns evaluationId	`201 Created`	
3	promote records decision successfully	`200 OK`	
4	close_evaluation seals the window	`200 OK`	

5	promote after close returns 409 - confirms append-only enforcement	`409 Conflict`	
6	generate_artifact returns Artifact 5 bundle	`200 OK + JSON`	
7	manifestRootHash present in bundle	`SHA-256 hash`	
8	KMS-backed signature present in environments where KMS signing is enabled	`RSA signature`	
9	Evaluation visible in dashboard under your project (production only)	`Visible`	
10	No cross-tenant data leakage confirmed - account team verifies (production only)	`Isolated`	

> NOTE: If steps 1 through 7 pass, your integration is complete. Contact your account team to provision your production key and confirm steps 8 through 10.

SECTION 12

Frequently Asked Questions

Do I need to change my database?

No. ProjectCore is additive. Your existing database, schema, and application logic remain untouched. It sits alongside your application as a sidecar API.

Can I reopen a closed evaluation?

No. Once closed, the evaluation is permanently sealed. This is the append-only guarantee. If you need to record additional decisions, open a new evaluation.

Why can't I just use my existing logs?

Logs describe activity. The ledger records responsibility. Logs are mutable - they can be altered, rotated, or deleted. The ledger is sealed with a cryptographic hash and independently verifiable by anyone with the public key. If a regulator or legal team asks who authorized a specific decision on a specific date, a log file cannot answer that question with forensic confidence. A sealed artifact can.

What languages are supported?

The API is standard REST/JSON and can be integrated from any language. A minimal TypeScript helper exists in the ProjectCore repo. Public generated SDKs are not yet shipped.

What is truthPosture and do I have to use it?

truthPosture classifies how grounded an agent reply is. known: directly backed by recalled Core memory. inferred: reasoned from known facts or general knowledge. unknown: explicitly uncertain or missing information. When authorityMode is agent_autonomous and truthPosture is unknown, the promote request is rejected. For human_led and human_in_the_loop, it is optional but recommended.

Does this add latency?

Typical promote() latency is sub-100ms, negligible relative to agent reasoning time. Health check and create_evaluation are similarly fast. generate_artifact involves KMS signing and takes slightly longer but is called once per evaluation window, not per decision.

What happens if the service goes down?

Artifact 5 files remain verifiable offline. Agents may fail open or closed depending on your risk tolerance. Contact your account team for SLA details.

Is Artifact 5 proprietary?

No. Standard JSON with SHA-256 verification and KMS signature. You own your evidence. You can verify it independently with the public key, forever, without access to ProjectCore infrastructure.

What if my API key is compromised?

Contact your account team immediately. Keys are revocable via status flag and a new key can be issued without affecting your existing sealed records. Sealed artifacts are not invalidated by key rotation.

What is the difference between evaluate() and agent-evaluations?

Decision sealing evaluations (POST /api/v1/evaluations) are bounded windows for recording and sealing AI-assisted decisions - the core four-call lifecycle. Agent evaluations (POST /api/v1/agent-evaluations) are structured assessments of individual agent behavior with scoring and results. If you are doing governance and compliance recording, use decision sealing evaluations.

SECTION 13

Next Steps

Follow this sequence from credentials to a verified production integration.

Step	Action	Notes
1	Confirm your sk_p0_ key is registered	Contact your account team before writing any code
2	Configure your .env file	Use the credentials provided during onboarding
3	Run the health check	health_check() should return ok: true
4	Add the five method calls	At your decision point in your application
5	Run the validation checklist	Steps 1 through 7 must pass
6	Verify your first Artifact 5	Confirm manifestRootHash and KMS signature are present
7	Confirm production auth and artifact-signing configuration	Contact your account team to verify auth posture and KMS configuration

---

Integration questions: **hello@project0.io**

Powerful AI is inevitable. Trustworthy AI is not - unless someone builds for it.