Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 November 2026

C. Rehfeld

27 May 2026

APIX Services Profile: Discovery Infrastructure for Web API and Bot Services
draft-rehfeld-apix-services-01

Abstract

This document defines the APIX Services Profile: an extension to the APIX Core Infrastructure specification that enables discovery and automated verification of web API and bot-consumable services. It specifies the APM field set for API services, the APIX Spider verification protocol, liveness monitoring configuration, search and filter query semantics, and the service record schemas (Level 1 summary and Level 2 full record) returned to consuming agents. Autonomous agents that implement this profile can discover API services globally from a single entry point, evaluate their trust posture without any out-of-band knowledge, and select services that satisfy their own Trust Policy.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 November 2026.

Table of Contents

1. Introduction

The APIX Core Infrastructure [APIX-CORE] defines the common foundation for agent-oriented service discovery: the governance model, trust model dimensions, commercial onboarding, sanctions compliance, and the base Index API. That document is the authoritative reference for all concepts and normative requirements shared across service types.

This document extends APIX Core for web API and bot-consumable services: services that expose a stable HTTPS endpoint, publish a machine-readable specification document, and are verifiable by an automated crawler. These are the primary service type for which APIX was originally conceived. The defining characteristics of an API service, as distinct from an IoT device service, are:

* A stable, publicly accessible entry_point URL.

* A machine-readable specification at a public spec.url (OpenAPI, MCP, AsyncAPI, or GraphQL SDL).

* Liveness maintained by Spider health checks, not device presence signals.

* No instance layer: the service is the service; there is no per-physical-unit record.

Implementors MUST satisfy all normative requirements in [APIX-CORE] before applying the additional requirements in this document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.1. Terminology

The following terms are defined in [APIX-CORE] and used here without redefinition: APIX, APM, APIX Manifest, service_id, trust model, organisation trust level (O-0 through O-5), service verification level (S-0 through S-4), liveness, commercial contract, sanctions screening.

Additional terms defined in this document:

API Service — A service registered with spec.type set to a web-protocol value (openapi, mcp, asyncapi, graphql). Has a stable entry_point URL and a machine-readable spec document. Verified by the APIX Spider.

APIX Spider — The automated crawler component that verifies API service registrations by fetching health endpoints and specification documents, comparing them against registered snapshots, and updating service verification levels.

Registered Spec Snapshot — The machine-readable specification document fetched and stored by the Spider at first registration, or at each APM update that increments api_version. The snapshot is the reference for structural consistency checks on subsequent Spider runs.

Liveness Monitoring Configuration — The per-service setting that determines how frequently the Spider rechecks a service’s health endpoint and specification document.

Liveness — Defined in [APIX-CORE] Section 2. For API services, liveness is measured by the APIX Spider via periodic HTTPS health checks against {entry_point}/health. The Spider records response availability, response time, and uptime percentage. Device service liveness uses a different mechanism (see [APIX-IOT]).

Service Record — The APIX-maintained record for an API service, combining the APM submitted by the Service Owner with Spider-verified trust metadata. Available at two granularity levels: Level 1 (summary, returned in search results) and Level 2 (full record, returned on detail fetch).

Lifecycle Stage — The current development and support status of a service: experimental, beta, stable, deprecated, or sunset.

spec_consistency — A Spider-maintained field with three values: consistent (spec matches snapshot), mismatch (spec changed without APM version update), unreachable (spec URL not fetchable or parseable).

Accredited Verifier — An organisation accredited by the governing body to perform O-4 and O-5 organisation trust assessments.

1.2. APM for API Services

1.2.1. Required Fields

The APIX Manifest for an API service extends the base APM format defined in [APIX-CORE]. The following complete schema applies to all services with spec.type in the Protocol Type Registry (Section 4):

```
{
  "apm_version": "1.0",
  "service_id": "unique stable ID -- UUID v4 or APIX-issued",
  "name": "human-readable service name",
  "description": "machine-parseable capability summary",
  "language": ["en"],
  "api_version": "semantic version string -- e.g. 2.1.0",
  "lifecycle_stage": "experimental|beta|stable|deprecated|sunset",
  "supersedes": "service_id this entry supersedes -- OPTIONAL",
  "owner": {
    "organisation_name": "legal entity name",
    "jurisdiction": "ISO 3166-1 alpha-2 country code",
    "registration_number": "reg. number -- required for O-2+",
    "contacts": {
      "operations": "email -- incident and spec-failure alerts",
      "escalation": "email -- Cluster 3 escalation; OPTIONAL"
    }
  },
  "spec": {
    "type": "value from protocols registry",
    "url": "URL to the machine-readable specification document",
    "version": "spec version string"
  },
  "capabilities": [
    "term from capabilities registry",
    "term from capabilities registry"
  ],
  "entry_point": "base URL of the service",
  "trust": {
    "organisation_level": "O-0 to O-5 -- set by index only",
    "service_level": "S-0 to S-4 -- set by index only",
    "spec_consistency": "null|consistent|mismatch|unreachable",
```

```
      "spec_fetch_consecutive_failures": 0,
      "next_spider_run_at": "ISO 8601 timestamp -- next Spider run",
      "liveness": {
        "last_ping_at": "ISO 8601 timestamp",
        "ping_interval_seconds": 300,
        "uptime_30d_percent": 99.9,
        "avg_response_ms": 142.0
      }
    },
    "notifications": {
      "supported": true,
      "channels": [
        {
          "type": "value from notification-channels registry",
          "registration_url": "URL to register a subscription",
          "events": ["event-type"],
          "spec_url": "URL to event schema -- OPTIONAL"
        }
      ]
    },
    "legal": {
      "terms_of_service_url": "URL",
      "privacy_policy_url": "URL",
      "data_processing_agreement_url": "URL -- required for O-3+",
      "gdpr_applicable": true,
      "jurisdiction_flags": ["ISO 3166-1 alpha-2 country code"]
    },
    "authentication": {
      "methods": ["oauth2 | api_key | bearer | mtls | none"],
      "oauth2_discovery_url": "URL to OAuth2 discovery endpoint"
    },
    "pricing": {
      "model": "free | freemium | paid | enterprise | dynamic",
      "pricing_url": "URL to human-readable pricing page -- OPTIONAL",
      "pricing_endpoint": "machine-readable price endpoint -- dynamic"
    },
    "standard_warnings": [
      {
        "field": "APM field path",
        "value": "the deprecated value in use",
        "registry_status": "deprecated",
        "deprecated_in_apix_version": "version string",
        "sunset_date": "ISO 8601 date",
        "replacement": "replacement value",
        "message": "human-readable warning"
      }
    ],
    "custom": [
```

```
      "com.example.coverage_polygon",
      "com.example.seasonal_availability"
    ]
  }
```

Field notes:

owner.contacts.operations MUST be provided. It is the primary notification address for all automated Spider alerts: spec fetch failures at Cluster 2 entry, liveness degradation, and recovery confirmations. This address SHOULD reach the team responsible for keeping the service registration current.

owner.contacts.escalation is OPTIONAL but RECOMMENDED. It is the escalation address sent to when failures reach Cluster 3 — indicating a persistent problem that has not been resolved through the Cluster 1 and Cluster 2 retry windows and likely requires management attention or a deliberate APM configuration update. This address SHOULD reach a team lead, service owner, or on-call manager. It MUST NOT be identical to operations — if the same person handles both, the escalation path provides no additional coverage.

api_version MUST follow Semantic Versioning 2.0.0. It describes the version of the service's own API, not the APM format version.

Each api_version value is bound to exactly one registered spec snapshot and carries an immutable field structure definition. The schema associated with a published api_version MUST NOT change after first publication:

* Adding a field requires a new minor version (e.g., 2.3.0 → 2.4.0).

* Removing a field, changing a field type, or making any other breaking change requires a new major version (e.g., 2.4.0 → 3.0.0).

This is not a convention — it is a hard invariant enforced by the snapshot model. A consuming agent that targets v2.4 receives a permanent contract: every service matching that version will expose exactly the fields defined for v2.4, no more and no less, for the lifetime of that registration. Service Owners are therefore free to drop outdated fields at a major version boundary without deprecation gymnastics — the major bump is the explicit, sufficient notice to consumers.

A Service Owner who modifies the live spec document at spec.url without submitting an APM update with a new api_version value WILL produce a structural mismatch between the live document and the

stored snapshot. The Spider MUST record this as an S-2 consistency failure and MUST surface it in the Service Record as a standard_warnings entry.

Operators who need to change their API MUST register a new api_version. This protects consuming agents from silent contract breakage.

language is OPTIONAL. When present, it MUST be a non-empty JSON array of BCP 47 language tags ([RFC5646]). It declares the natural languages in which the service's API responses, documentation, and user-facing content are available. If omitted, the default value is ["en"]. Agents MAY use this field to select services that operate in a required language. Example: ["de", "en"] for a service supporting German and English.

authentication is OPTIONAL but RECOMMENDED. When present, it MUST contain a methods array listing one or more of the following values: "oauth2", "api_key", "bearer", "mtls", "none". A service that requires no authentication MUST declare ["none"] explicitly. When "oauth2" is in methods, oauth2_discovery_url MUST be provided; it MUST point to a valid OAuth2 discovery document (RFC 8414 or OpenID Connect Discovery). The authentication field enables agents to pre-qualify whether they can authenticate with a service before invocation, avoiding wasted round-trips and quota consumption against services the agent cannot use. The APIX Spider does not verify credential validity; it verifies only that oauth2_discovery_url, when declared, is reachable and returns a parseable discovery document.

pricing is OPTIONAL. When present, it MUST contain a model field with one of the following values:

* "free" — no charge for any usage; pricing_url MAY be omitted.

* "freemium" — a free tier exists; pricing_url SHOULD be provided.

* "paid" — all usage is charged; pricing_url MUST be provided.

* "enterprise" — pricing is individually negotiated; pricing_url MUST be provided.

* "dynamic" — price varies based on service load or other real-time factors; pricing_endpoint MUST be provided.

pricing.model is self-declared and is not verified by the APIX Spider. The index stores and exposes only the declared model value and the pricing_endpoint URL; it does not fetch, cache, or display current prices. Consuming agents are solely responsible for querying

the pricing_endpoint directly before invocation and for evaluating the returned price against their operator-configured budget constraints. Misrepresentation of the pricing model (e.g., declaring "free" while charging) constitutes a breach of the service owner's registration contract and MUST result in suspension at O-3+.

pricing.pricing_url MUST be a stable HTTPS URL pointing to a human-readable pricing page. Agents MAY follow this URL for full pricing detail.

pricing.pricing_endpoint is REQUIRED when model is "dynamic" and MUST NOT be present for any other model value. It is a stable HTTPS URL that agents query immediately before service invocation to obtain the current price. The endpoint MUST respond with:

```
{
  "price_per_unit": 0.005,
  "currency": "ISO 4217 currency code -- e.g. USD, EUR",
  "unit": "request | token | minute | kb | device_hour",
  "valid_until": "ISO 8601 timestamp -- quote expires at this time",
  "quote_id": "single-use price reservation token",
  "load_factor": 1.2
}
```

quote_id is a single-use price reservation token issued by the service for this specific price quote. It MUST be treated as a binding commitment: the service MUST honour the quoted price_per_unit when the agent presents a valid, non-expired quote_id at invocation time. The quote_id expires at valid_until and MUST NOT be accepted after that timestamp. A quote_id is single-use — once presented in an invocation it is consumed and MUST NOT be accepted again. How the agent presents the quote_id during invocation (HTTP header, request body field, or other mechanism) is defined by the service's own API documentation; APIX does not specify the invocation protocol.

An agent MUST NOT invoke the service after valid_until without re-querying the pricing endpoint to obtain a fresh quote_id. An agent MAY compare price_per_unit against its operator-configured budget ceiling and MUST abandon the invocation and discard the quote_id if the ceiling is exceeded.

load_factor is OPTIONAL and indicates the current multiplier relative to the service's baseline price (1.0 = baseline).

lifecycle_stage defines the publication maturity of this API service. API service lifecycle is defined by this profile. Valid values are experimental, beta, stable, deprecated, and sunset. Default if omitted is stable. Services at experimental or beta are excluded from default search results (see Section 7.2).

Stage	Meaning
experimental	Pre-release. Interface may change without notice.
beta	Feature-complete but not yet declared stable. Breaking changes possible.
stable	Production-ready. Breaking changes require a new api_version.
deprecated	Service owner is winding down. Successor SHOULD be declared via supersedes.
sunset	Service endpoint will go dark at the declared sunset_date. Agents MUST stop using the service by that date.

Table 1

Transitions are unidirectional: experimental → beta → stable → deprecated → sunset.

supersedes is OPTIONAL. When set, the index MUST automatically set superseded_by on the referenced entry. The referencing service MUST be registered under the same organisation account.

trust fields are set exclusively by the index operator based on verification outcomes. APM submissions that include trust field values MUST have those values overwritten by the index upon processing.

standard_warnings is set exclusively by the index operator. It is populated only after the grace period for the relevant deprecation has elapsed (see [APIX-CORE] Section 4.3). During the grace period the field MUST be empty even if the service uses a deprecated value. Service Owners MUST NOT submit this field; submitted values MUST be ignored.

custom is OPTIONAL. When present, it MUST be a JSON array of strings. Each string MUST use reverse-domain notation (e.g., "com.example.coverage_polygon") to prevent key collisions between independent adopters. The array MUST NOT contain more than 20 entries. Each entry MUST NOT exceed 128 characters.

The custom array is a capability declaration list — it signals that the service exposes properties not yet covered by the standard schema, without encoding their values in the index. Values associated with declared custom properties are defined and published by the registrant outside the index (e.g., in their API documentation or spec document) and are retrieved directly from the service. The index stores and exposes only the declared key names.

The custom field is the governed extension mechanism for early adoption of field patterns that may be standardised in a future APM version. The APIX Spider records declared key names in the index's custom key registry. Consuming agents can discover services that declare a known custom property via the custom_key search parameter; interpretation of a custom property's semantics requires out-of-band agreement between registrant and consumer.

The promotion path: when a custom key appears in APM submissions from five (5) or more independent organisations, The governing body MAY open a governance track to evaluate the pattern as a candidate standard field in a future APM version. Registrants are encouraged to document custom property semantics publicly to support interoperability and accelerate standardisation.

notifications is OPTIONAL for experimental and beta lifecycle stages and RECOMMENDED for stable. If notifications.supported is true, notifications.channels MUST contain at least one entry.

entry_point is the base HTTPS URL of the service, used by consuming agents to construct API calls. The following normative requirements apply:

* entry_point MUST use the https scheme. HTTP entry points MUST be rejected at registration.

* entry_point MUST remain stable for the lifetime of the service registration. A change to entry_point MUST be submitted as an APM update and MUST trigger immediate Spider re-verification.

* The Spider MUST NOT hit entry_point directly for liveness checks. Instead, the Spider checks entry_point + /health (see Section 5.2).

* HTTP redirects from entry_point are permitted for consuming agents but MUST NOT be present at entry_point/health (the health endpoint MUST respond directly without redirect).

entry_point/health is the mandatory liveness endpoint. Every registered service MUST expose a health endpoint at the path /health relative to entry_point. This endpoint:

* MUST return HTTP 2xx when the service is operational.

* MUST return without requiring authentication.

* MUST respond within a reasonable timeout (RECOMMENDED: 5 seconds).

* SHOULD return a JSON body of the form {"status": "ok", "api_version": "<semver>"}. If api_version is present, the Spider SHOULD cross-check it against the APM api_version field; a mismatch MUST be recorded as a warning in the Service Record.

* MUST NOT be used by consuming agents for API calls — it is a Spider-facing infrastructure endpoint only.

spec.url is the URL to the machine-readable API specification document (OpenAPI JSON/YAML, MCP manifest, AsyncAPI document, or GraphQL SDL).

* spec.url MUST use the https scheme.

* spec.url MUST be publicly accessible without authentication. A spec behind authentication cannot be fetched by the Spider and permanently prevents the service from reaching S-2.

* On the initial Spider run following registration, the Spider fetches the spec document and stores it as the registered spec snapshot. All subsequent Spider runs compare the live document at spec.url against this snapshot to detect breaking changes (S-3 assessment). The snapshot is updated when the Service Owner submits an APM update that increments api_version.

* An APM update that changes spec.url MUST trigger immediate Spider re-verification and snapshot replacement (see Section 5.1).

The service_id MUST be stable across re-registrations and version updates. It is the canonical identity of the service in the APIX and MUST be a UUID v4 or an APIX-issued deterministic identifier.

1.3. Protocol Type Registry (v1.0 starter set)

The spec.type field MUST contain a value from the Protocol Type Registry at apix.example.org/registry/protocols. The registry is the authoritative and always-current list of valid values. The entries below are the v1.0 starter set for API services; the governing body extends the registry as additional protocol types reach sufficient adoption. Registry entries follow the lifecycle defined in [APIX-CORE].

Registry value	Standard	Spider behaviour	Status
openapi	OpenAPI 3.x	Parses paths, schemas, auth requirements	active
mcp	Model Context Protocol	Parses tool definitions and capability list	active
asyncapi	AsyncAPI 2.x / 3.x	Parses channels, message schemas	active
graphql	GraphQL SDL	Introspection query to entry_point; see below	active

Table 2

Note: IoT device service types (device-class, hub) are defined in [APIX-IOT] and are not governed by this profile.

1.3.1. GraphQL Spider Behaviour

For services with spec.type: graphql, the spec.url field MUST point to the GraphQL endpoint (identical to entry_point in most deployments). The Spider MUST issue a standard GraphQL introspection query to that endpoint:

```
{ __schema {
    queryType { name }
    mutationType { name }
    subscriptionType { name }
    types {
      name
      kind
      fields(includeDeprecated: true) {
        name
        isDeprecated
        type { name kind ofType { name kind } }
        args { name type { name kind ofType { name kind } } }
      }
    }
} }
```

The Spider MUST POST this query as Content-Type: application/json with {"query": "<introspection query>"}. A 200 response with a valid JSON body containing a data.__schema object constitutes a successful spec fetch (S-2 prerequisite).

Normalised schema representation: The Spider MUST normalise the introspection result into a canonical form before storage as the registered spec snapshot. The canonical form is a sorted JSON object containing:

* types: alphabetically sorted list of non-introspection type names (excluding built-in scalar and introspection types: __Schema, __Type, __TypeKind, __Field, __InputValue, __EnumValue, __Directive, __DirectiveLocation, Boolean, Int, Float, String, ID)

* Per type: kind, sorted fields list, per field: name, required/optional status, argument names and types, return type name

Normalisation eliminates cosmetic differences (field order, whitespace) from breaking change detection.

Breaking change definition for GraphQL:

Change	Breaking
Removed type	Yes
Removed field on existing type	Yes
Added required argument to existing field	Yes
Changed return type of existing field	Yes
Added optional field	No
Added new type	No
Added optional argument	No
Deprecated field (marked isDeprecated: true)	No — recorded as metadata warning only

Table 3

Introspection disabled: Some production GraphQL services disable introspection as a security measure. If the introspection query returns an error (HTTP 200 with {"errors": [...]} containing a message indicating introspection is disabled, or HTTP 4xx), the Spider MUST set spec_consistency: unreachable and record a metadata warning with code graphql_introspection_disabled. The service CANNOT achieve S-2 without enabling introspection for the Spider's IP range, or by providing an alternative static SDL document at a dedicated spec.url (in which case spec.type MUST be graphql and the Spider will fetch and parse the SDL document directly instead of issuing the introspection query).

Services whose specification type is not yet in the registry SHOULD request addition via the governing body's registry extension process before registering. Until the type is added, such services cannot achieve S-2 or above, as the Spider has no parser for unregistered types.

1.4. Capability Taxonomy Registry (v1.0 starter set)

The capabilities field MUST contain terms from the Capability Taxonomy Registry at apix.example.org/registry/capabilities. The registry is the authoritative and always-current list of valid terms. Terms are hierarchical, dot-separated (e.g., commerce.marketplace), and follow the lifecycle defined in [APIX-CORE].

The following are the v1.0 starter set for API services. IoT-specific capability terms (iot, home.*) are defined in [APIX-IOT]. The live registry is the authoritative source; this list is illustrative only.

Term	Description	Status
commerce	E-commerce and marketplaces	active
commerce.marketplace	Multi-vendor marketplace	active
commerce.retail	Single-vendor retail	active
payments	Payment processing	active
payments.card	Card payment processing	active
payments.crypto	Cryptocurrency payments	active
data.financial	Financial data and markets	active
data.legal	Legal documents and data	active
nlp	Natural language processing	active
nlp.translation	Language translation	active
identity	Authentication and identity	active
communication	Messaging and notifications	active
storage	File and object storage	active
compute	Code execution and computing	active
media	Image, audio, video services	active
search	Information retrieval	active

Table 4

A Service MUST declare at least one capability term. Declared capabilities are validated by the Spider against the parsed specification where the spec type supports it. Services using deprecated taxonomy terms receive a standard_warnings entry in their Service Record.

1.4.1. Hierarchical semantics

Capability terms form a strict containment hierarchy expressed by their dot-separated segments. A term is a _sub-capability_ of every term obtained by removing one or more trailing segments: payments.card is a sub-capability of payments; logistics.tracking.express is a sub-capability of both logistics.tracking and logistics. Equivalently, declaring a service under a term asserts that the service is an instance of every ancestor term (is_instance_of).

Containment MUST be evaluated on whole dot-separated segments, never on raw character prefixes. A term T contains a term U if and only if U equals T, or U begins with T immediately followed by a . separator. Thus payments contains payments.card but does not contain a distinct term that merely shares a leading substring.

A service that declares a sub-capability is discoverable under every ancestor term without separately declaring the ancestors; ancestor expansion is performed by the index, not by the registrant. This hierarchy is the basis for hierarchical capability search (see the capability search parameter) and for graph-based discovery.

1.4.2. Capability Taxonomy Governance

The Capability Taxonomy Registry is maintained by the governing body. The following process governs additions, deprecations, and removals:

Proposing new terms:

Any organisation may propose a new taxonomy term. A proposal MUST include: a candidate term (dot-separated, max 4 levels), a definition (max 200 words), examples of services that would use it, and evidence of adoption (at minimum 3 registered or planned APIX services that require the term). Proposals MUST be submitted via the governing body’s designated public proposal mechanism.

Review cycle: the governing body reviews taxonomy proposals quarterly. A proposal is accepted when: the definition is unambiguous, the term does not overlap with an existing term, and the adoption evidence is credible. Accepted terms are added as active in the next quarterly registry update.

Versioning: The Capability Taxonomy Registry is independently versioned. The registry version is exposed in the APIX root resource (see [APIX-CORE] Section 9.2). The taxonomy version advances on any addition or deprecation.

Deprecation: A term is deprecated when a more precise replacement term is accepted. Deprecation follows the lifecycle defined in [APIX-CORE] Section 4.3: 12-month minimum window, 90-day grace period.

IANA considerations: The taxonomy is not maintained by IANA. The governing body is the designated authority. Should a future version of this specification transfer maintenance to IANA, a mapping document MUST be published.

1.5. Notification Channel Registry (v1.0 starter set)

The notifications.channels[].type field MUST contain a value from the Notification Channel Registry at apix.example.org/registry/notification-channels. The following are the v1.0 starter set. Registry entries follow the lifecycle defined in [APIX-CORE].

Registry value	Transport	Direction	Applicable to	Semantics
webhook	HTTPS POST	Server → agent	All service types	Service posts event payloads to an agent-registered callback URL. Agent provides the callback URL at subscription time via the registration_url endpoint. The service MUST include a shared secret in each delivery; agents MUST verify it. At-least-once delivery; agents MUST be idempotent.
sse	HTTP Server-Sent Events	Server → agent	API services only	Agent opens a long-lived HTTP GET connection to the service's event stream. No subscription

				management endpoint required; the registration_url field carries the SSE stream URL. Agent reconnects on disconnect per the SSE specification ([W3C-SSE]).
websocket	WebSocket (RFC 6455)	Bidirectional	API services only	Agent opens a WebSocket connection to the service. registration_url is the WebSocket endpoint. Enables request/response and push notification in a single persistent connection.

Table 5

Subscription management: For webhook, the registration_url MUST point to an HTTPS endpoint that accepts a subscription registration payload. The minimum subscription registration format is:

```
{
  "callback_url": "https://agent.example/events",
  "events": ["spec.updated", "status.changed"],
  "secret": "agent-supplied shared secret for HMAC verification"
}
```

The registration endpoint MUST return the subscription identifier and expiry time. Agents are responsible for renewing subscriptions before expiry.

Event types: The events field in the APM notification channel and in subscription requests MUST contain values from the APIX Event Type Registry at apix.example.org/registry/event-types. The v1.0 starter set:

Event type	Description
spec.updated	The service's spec document has changed (new api_version)
status.changed	Service operational status has changed (up/down/degraded)
trust.changed	Organisation or service trust level has changed

Table 6

Notification channel requirements are OPTIONAL for registration. Services that do not support push notifications MUST set notifications.supported to false. Consuming agents that require push notifications SHOULD filter by notifications.supported: true before subscribing.

1.6. Registration and Onboarding

1.6.1. Push Registration (Human-Initiated)

Service registration MUST be human-initiated. The registrant MUST agree to the index operator's Terms of Service before any service record is activated. For O-0 and O-1, self-service portal registration with accepted Terms of Service satisfies this requirement. For O-2 and above, registration MUST additionally involve a formal B2B contractual relationship between the Service Owner and the index operator or its Accredited Regional Representative.

Registration MUST be scoped at the *organisation level*. An organisation registers once and undergoes identity verification once; multiple services may then be registered under that organisational identity. This requirement ensures:

* Identity verification and sanctions screening are performed once per legal entity, not repeated per service.

* Organisation trust (O-level) established at registration propagates to all services registered under that organisation without re-verification of the organisation's identity.

Definition: one service equals one APIX Manifest (APM) document with one distinct entry_point. Logical bundling of API paths under a single entry point is the registrant's responsibility and is permitted.

The registration process:

1. The Service Owner (or their Accredited Regional Representative) creates an Organisation Account in the APIX Registration Portal. The index operator MUST screen the Service Owner against applicable sanctions lists before account activation per [APIX-CORE].

2. The Service Owner provides organisation details sufficient for the target Organisation trust level. This step is performed once per organisation.

3. The Service Owner submits an APIX Manifest for each service to be registered, including the spec URL and entry point. Each service is associated with a liveness monitoring configuration that determines Spider check frequency (see Section 5.3).

4. For O-1: email and domain ownership verification is completed automatically.

5. For O-2: the index operator or Regional Representative verifies the declared company registration number.

6. For O-3: APIX automatically checks security.txt, DMARC/SPF records, and declared legal document URLs. No human review required.

7. For O-4 and O-5: the Service Owner engages an Accredited Verifier (O-4) or submits an audit certificate directly to the governing body (O-5).

8. Upon completion of applicable checks, the service is activated in the index and the Spider is triggered.

1.6.2. Accredited Verifier Requirements

Organisation levels O-4 and O-5 require an Accredited Verifier. To be accredited by the governing body, a candidate Verifier organisation MUST satisfy all of the following:

Independence: The Verifier MUST have no ownership relationship, employment relationship, or consulting engagement with any organisation it assesses. Independence is evaluated per assessment: a Verifier that provided consulting services to the candidate organisation within the prior 24 months MUST recuse itself from that assessment and the governing body MUST assign an alternate Verifier.

Demonstrated audit competence: The Verifier organisation MUST hold at least one of: ISO/IEC 27001 Lead Auditor certification (per ISO/IEC 17021), SOC 2 attestation authority (AICPA CPA firm licensed for attestation engagements), or ISAE 3402 Type II authority. At least one named individual from the Verifier organisation MUST hold a current relevant professional certification (CISA, CISSP, or equivalent recognised by the governing body).

APIX trust level: The Verifier organisation MUST itself be registered in APIX at O-2 or above.

Contractual obligation: The Verifier MUST sign the Verifier Agreement, which binds it to: the Verifier Standard, liability for negligent assessments, incident reporting obligations, and annual re-accreditation.

Annual review: Accreditation is reviewed annually by the governing body. Failure to maintain the requirements above or material error in an assessment MUST result in suspension pending review. The governing body MUST maintain a public registry of all accredited Verifiers and their current accreditation status.

Revocation: the governing body MAY revoke accreditation at any time for material breach of the Verifier Agreement, demonstrated conflict of interest, or failure to maintain competence requirements. Revocation is immediate; pending assessments are transferred to an alternate Verifier at no additional cost to the assessed organisation.

1.6.3. Spider Verification (Automated)

The APIX Spider is triggered automatically at two points:

* *At registration*: once a service is activated, the Spider performs an initial verification run to establish the baseline Service Verification Level.

* *On schedule*: thereafter, the Spider rechecks the service at the interval defined by the service’s commercial tier (see Section 5.3).

During a Spider run, the Spider:

1. Performs an HTTPS request to {entry_point}/health and records the response code, response time, and timestamp (Liveness: S-1).

2. Fetches the spec document from spec.url (HTTPS, no authentication).

3. Parses the fetched document and compares it structurally against the registered spec snapshot (S-2 if fetchable and consistent; S-3 assessed if no breaking changes detected across three or more consecutive runs).

4. Updates all Liveness metrics in the Service Record.

5. Records any failures and increments consecutive_failures.

6. If the APM contains a custom field, records each declared key name in the index's custom key registry. No values are stored; the custom array contains only key name strings.

The Spider MUST NOT call any API endpoint beyond {entry_point}/health and spec.url. The Spider MUST NOT submit data to, create resources in, or otherwise interact with the production API of a registered service.

The Spider MUST respect HTTP rate limits declared by the service. Spider requests MUST include a User-Agent header identifying the APIX Spider and version.

1.6.4. Service Verification Level Ceilings

Certain service configuration states and governing body actions create hard ceilings on the maximum Service Verification Level achievable or maintainable. The following table defines all ceiling conditions for API services. The index MUST enforce these ceilings automatically.

Condition	Maximum S-level
spec.url requires authentication	S-1
spec.url unreachable or unparseable	S-1
Spec type not in APIX Protocol Type Registry	S-1
GraphQL service with introspection disabled and no SDL at spec.url	S-1
Fewer than three consecutive Spider runs completed	S-2
Liveness frequency set to initial (Spider runs once only)	S-2
Active security_incident flag set by governing body	S-2
No completed security scan or pen test certificate on file	S-3

Table 7

security_incident is an index-maintained flag, not an APM field. It is set by the governing body upon receipt of credible breach intelligence and cleared only after the service owner demonstrates full remediation and passes a new security scan. The process for setting, contesting, and clearing this flag — including the evidence threshold, contestation procedure, and integration with external security feeds — is an open question (see Section Open Questions).

1.6.5. Free Registration Tier and Abuse Deterrence

Service registration at O-0 and O-1 trust levels constitutes the free registration tier. No payment information is required for these levels. This is a deliberate design choice: requiring payment for O-0 would create a barrier to early adoption and conflict with the open infrastructure mission.

The following controls are sufficient at the free tier:

1. *Mandatory Terms of Service acceptance:* All registrations — including free tier — require agreement to the index operator's Terms of Service before account activation. For O-0 and O-1,

this self-service acceptance is the sole contractual requirement and creates legal accountability regardless of tier. For O-2 and above, a formal B2B contract is additionally required.

2. *Sanctions screening at activation:* All organisations are screened before account activation per [APIX-CORE] Section 7. Sanctioned entities cannot register at any tier.

3. *Default discovery exclusion:* Consuming agents applying standard Trust Policies (org_level_min O-1 or higher) structurally exclude O-0 services from results. Abuse at O-0 cannot reach consuming agents that apply minimum recommended trust filters. Consuming agents that accept O-0 services MUST do so deliberately.

4. *Liveness gating:* O-0 services configured at initial liveness frequency are excluded from default search results by default (see Section 5.3). An abusive operator that registers a fake service and then takes it offline will be gated out by both trust filtering and liveness filtering simultaneously.

Payment information on file is NOT required for O-0 or O-1 registration. A future version of this specification MAY require payment information on file (without charge) for O-1 as an additional identity anchor if operational experience shows the above controls insufficient. This is not normative for the current version.

1.6.6. Liveness Monitoring Configuration

Each registered API service MUST have a liveness monitoring configuration that determines Spider check frequency. This configuration:

* Is set per service, not per organisation account. An organisation MAY configure different check frequencies for different services registered under the same account.

* MUST be agreed in the commercial contract between the Service Owner and the index operator.

* Determines the maximum age of last_ping_at data available to consuming agents for that service.

Implementations MUST support at minimum the following frequency classes, identified by their normative spider_interval value in the Service Record:

Frequency class	Maximum spider_interval	Normative label
Initial only	N/A (one run at activation)	"initial"
Daily	86,400 seconds	"daily"
Hourly	3,600 seconds	"hourly"
High-frequency	300 seconds	"high"

Table 8

Implementations MAY define additional frequency classes. The spider_interval field in the Service Record MUST reflect the actual configured interval in seconds.

Effect on trust signal quality: A consuming agent applying a last_ping_age < N filter will structurally exclude services whose check frequency cannot produce sufficiently fresh liveness data. Liveness monitoring configuration is therefore a market signal: services requiring discovery by latency-sensitive agents must invest in check frequency sufficient to satisfy those agents' trust policies.

Services configured at initial-only frequency MUST be excluded from standard discovery query results by default. Consuming agents MUST explicitly opt in to include initial-only services in result sets.

1.7. Spider and Crawler Specification

1.7.1. Crawl Triggers

The Spider is triggered by the following events:

Trigger	Description
Registration activation	Immediate first run when a service is activated
Scheduled interval	Recurring, per service liveness monitoring configuration (Section 5.3)
Manual re-trigger	Service Owner may request a manual re-trigger once per 24 hours
Spec URL change	An APM update that changes the spec.url triggers an immediate run

Table 9

1.7.2. Automated Verification Scope

The Spider performs the following checks in sequence. Each check's result is stored independently; a failure at one level does not prevent checks at other levels from being recorded.

The Spider MUST use HTTPS for all outbound requests. The Spider MUST NOT send authentication credentials to any registered service. Spider requests to entry_point/health or spec.url MUST NOT include Authorization headers, API keys, cookies, or client certificates.

If a request returns an HTTP redirect (3xx), the Spider MUST follow the redirect only if the redirect target also uses HTTPS. The Spider MUST NOT follow redirects from HTTPS to HTTP.

1. *Liveness check*: HTTPS GET to {entry_point}/health. Record HTTP status code, response time, and timestamp. A 2xx response without authentication constitutes a successful liveness check (S-1). If the response body is valid JSON containing an api_version field, the Spider MUST cross-check this value against the api_version declared in the APM. A mismatch is recorded as a metadata warning, not a liveness failure.

2. *Spec fetch*: HTTPS GET to spec.url. The Spider MUST NOT send authentication credentials. A successful fetch (2xx response, non-empty body) is the prerequisite for steps 3 and 4. Record content type and document size.

3. *Spec parse and consistency check*: Parse the fetched document according to the declared spec.type. Compare it structurally against the registered spec snapshot stored at initial registration time. The Spider MUST set spec_consistency to one of three values after every run:

   * consistent — document is fetchable, parseable, and structurally matches the registered snapshot. Constitutes S-2 verification.

   * mismatch — document is fetchable and parseable, but structurally differs from the snapshot (paths removed, required fields changed, response schemas changed). S-2 is revoked; standard_warnings is updated. This indicates operator-caused contract breakage.

   * unreachable — spec.url returned a non-2xx response, was not reachable, or the document could not be parsed. S-2 is not achieved or is suspended. This indicates an availability problem, not a contract violation. spec_consistency MUST be null only before the Spider's first run on a newly registered service. Once any run completes, the field MUST carry one of the three values above. The Spider MUST NOT call any API endpoint declared in the spec. Spec verification is document comparison only.

4. *Breaking change detection*: Compare the current parsed spec against the registered spec snapshot. Flag removed paths, changed required fields, or changed response schemas as breaking changes. Three or more consecutive runs with no breaking changes detected are required for S-3 verification.

5. *Liveness metrics update*: Update last_ping_at, uptime_30d_percent, avg_response_ms, consecutive_failures, and next_spider_run_at.

1.7.3. Failure Handling

Liveness failures (entry_point/health unreachable):

* A single failed ping increments consecutive_failures and updates last_ping_at with the failure timestamp.

* After 3 consecutive failures, the Service Record is flagged as status: degraded in the index.

* After 10 consecutive failures, the Service Record is flagged as status: unreachable and is excluded from standard search results.

* contacts.operations is notified at the 3-failure threshold (incident warning). Both contacts.operations and contacts.escalation are notified at the 10-failure threshold (service unreachable escalation).

* A service that recovers (next ping succeeds) has its status restored and consecutive_failures reset to 0 automatically.

Spec fetch failures (spec_consistency: unreachable):

Spec fetch failures have distinct probable causes depending on how long the failure persists. The Spider MUST apply a three-cluster retry model that targets the likely cause window at each stage. Cluster escalation is triggered by spec_fetch_consecutive_failures crossing a threshold.

Cluster 1 — Infrastructure / network (failures 13): Cause: transient network loss, host restart, or CDN hiccup. Retry: 5 min → 15 min → 30 min. No notification.

Cluster 2 — Application (failures 46): Cause: software instability (crash loop, OOM, startup failure). Retry: 2 h → 4 h → 8 h. Notification: email to contacts.operations at failure 4.

Cluster 3 — Configuration (failures 7+): Cause: persistent misconfiguration — wrong spec.url, auth gate added, URL moved. Retry: 24 h → 72 h (cap). Notification: email to contacts.operations AND contacts.escalation at failure 7.

* spec_consistency is set to unreachable on the first failure and remains unreachable until a successful fetch.

* next_spider_run_at is set to the next retry timestamp after each failed run so Service Owners can observe when the retry will occur.

* A successful spec fetch resets spec_fetch_consecutive_failures to 0 and sets spec_consistency to consistent or mismatch as appropriate.

* spec_fetch_consecutive_failures MUST be visible in the Service Record so Service Owners can monitor retry cluster state without contacting the Index operator.

Manual re-trigger:

The Index operator SHOULD provide a mechanism for Service Owners to request an immediate Spider re-run outside of the scheduled interval. This is the primary recovery mechanism when a service has been repaired and the operator does not want to wait for the next scheduled retry.

When a manual re-trigger is requested: - next_spider_run_at MUST be set to the current timestamp. - spec_fetch_consecutive_failures MUST be reset to 0, returning the service to Cluster 1 retry behaviour for the next run. - The Spider MUST execute the run as soon as scheduling permits.

The Index MAY rate-limit manual re-triggers to at most once per hour per service to prevent abuse.

1.8. Index API — Services Layer

1.8.1. Search and Filter API

The search endpoint applies server-side filters to reduce result sets before transmission. Only filters on indexed scalar values are server-side; filters requiring deep metadata evaluation are applied client-side after fetching the Level 2 Service Record (Section 7.4).

API version scoping (path segment):

The optional {api_version} path segment scopes the search to services whose api_version starts with the specified major or major.minor prefix. The segment uses the v prefix followed by the semver major or major.minor value (e.g., v2, v2.4). When absent, no version constraint is applied.

```
GET /search/v2/?capability=payments.card   ← v2.x.x services only
GET /search/v2.4/?q=payment               ← v2.4.x services only
GET /search/?q=payment                    ← no version constraint
```

The api_version immutability invariant (see [APIX-CORE]) means that a consuming agent pinned to /search/v2.4/ receives a stable, permanent schema contract: every result exposes exactly the fields defined for v2.4.

The index does not perform cross-version mapping. A service registered at v3.0 does not appear in /search/v2/ results and is never synthesised into a v2-shaped response. There are no null substitutions for dropped fields and no type coercions for changed fields across version boundaries. If a pinned-version query returns empty results, the agent SHOULD follow the superseded_by link in any previously known Level 2 record to discover the current version, or

issue GET /search/ with no path segment and no query parameters to retrieve the version landscape (see [APIX-CORE]). The parameter-less /search/ endpoint returns version metadata only and executes no content query — it is a discovery resource, not a full-index dump.

Example — buying bot querying for marketplace services:

```
GET /search/v2/?capability=commerce.marketplace
              &protocol=mcp,openapi
              &org_level_min=O-2
              &service_level_min=S-2
              &max_ping_age=3600
              &uptime_30d_min=95.0
              &lifecycle_stage=stable
              &page=1
              &page_size=20
```

Version landscape response (GET /search/ — no path segment, no query parameters):

When the search endpoint is called with no api_version path segment and no query parameters, it MUST return the version landscape and MUST NOT return service records:

```
{
  "version_landscape": [
    {
      "api_version_prefix": "v1",
      "service_count": 0,
      "lifecycle_status": "sunset"
    },
    {
      "api_version_prefix": "v2",
      "service_count": 1847,
      "lifecycle_status": "stable",
      "_links": {
        "search": {
          "href": "https://apix.example.org/search/v2/{?...}",
          "templated": true
        }
      }
    },
    {
      "api_version_prefix": "v3",
      "service_count": 4201,
      "lifecycle_status": "stable",
      "_links": {
        "search": {
          "href": "https://apix.example.org/search/v3/{?...}",
          "templated": true
        }
      }
    }
  ]
}
```

Consuming agents that receive empty results from a pinned-version query MUST use this endpoint to survey the current version landscape before re-issuing a content query. The _links.search template in each entry provides the correctly scoped search URL for that version prefix.

Normative server-side filter parameters:

Parameter	Type	Default	Description
q	string	—	Free-text search across name and description
capability	string	—	Capability taxonomy term. MUST be an active or

			deprecated registry value. Matched hierarchically by default — see "Hierarchical capability matching" below
capability_match	enum	subtree	Capability match mode: subtree returns the term and all its sub-capabilities; exact returns only services declaring the exact term
protocol	string	—	Comma-separated protocol type values. MUST be values from the Protocol Type Registry
org_level_min	enum	O-0	Minimum Organisation trust level. Excludes services below threshold
service_level_min	enum	S-0	Minimum Service verification level
max_ping_age	integer	—	Maximum seconds since last_ping_at. Excludes services with older liveness data
uptime_30d_min	float	—	Minimum 30-day uptime percentage
lifecycle_stage	enum	stable	Filter by lifecycle stage. Default excludes experimental, beta, deprecated, and sunset
include_superseded	boolean	false	When false, excludes services for which superseded_by is set. When true, all matching versions are returned
spec_consistency	enum	—	Filter by spec consistency status. Values: consistent, mismatch, unreachable. null (Spider not yet run) is excluded when any value is specified. When absent, no constraint is applied. Agents performing consequential tasks SHOULD explicitly pass

			consistent
language	string	—	BCP 47 language tag ([RFC5646]). Returns only services whose language array contains the specified tag. Services with no language field are treated as ["en"]
pricing_model	enum	—	Filter by declared pricing model. Values: free, freemium, paid, enterprise, dynamic. When absent, no pricing constraint is applied
auth_method	enum	—	Filter by supported authentication method. Values: oauth2, api_key, bearer, mtls, none. Returns services whose authentication.methods array contains the specified value
deployment_region	string	—	Cloud or jurisdiction region identifier (e.g. eu-west-1, eu). Returns only services that declare the specified region in deployment_regions. Candidate field — see Open Question 8
near	string	—	Decimal lat,lon coordinate pair (e.g. 48.137,11.576). When combined with coverage_radius_km, returns only services whose declared coverage area overlaps the specified point. Candidate field — see Open Question 8
coverage_radius_km	integer	—	Radius in kilometres around the near coordinate. Only meaningful when near is also specified. Candidate field — see Open Question 8
custom_key	string	—	Reverse-domain key name (e.g. com.example.coverage_polygon).

			Returns only services whose custom object contains the specified top-level key. Values are not searchable; key presence only
page	integer	1	Result page number
page_size	integer	20	Results per page. Maximum: 100

Table 10

All filter parameters are OPTIONAL. When absent, the parameter imposes no constraint except lifecycle_stage (default stable), include_superseded (default false), and capability_match (default subtree).

Hierarchical capability matching. A ?capability=T query MUST return every service whose declared capabilities array contains T or any sub-capability of T, using the whole-segment containment rule defined under "Hierarchical semantics" in the Capability Taxonomy Registry section. For example, ?capability=payments MUST return services that declared payments.card or payments.crypto, not only those that declared payments exactly. Setting capability_match=exact restricts the result to services declaring T exactly. Ancestor expansion is performed by the index at query time; a service need not declare ancestor terms to be discoverable under them.

Results are returned as paginated Level 1 Search Result records (Section 7.2) with HATEOAS links to Level 2 Service Records. Pagination is REQUIRED.

Design note — natural language search is out of scope for the base specification: The APIX search endpoint provides keyword and structured-filter search only. Semantic or natural-language query processing — where a free-text prompt such as "find me a low-latency European payment API that handles refunds" is resolved to a ranked service list by embedding or language model inference — is explicitly excluded from this version of the specification. The I-D draft-cui-ai-agent-discovery-invocation ([I-D.cui-ai-agent-discovery-invocation]) describes this pattern as an optional registry capability (MAY); APIX does not exercise this option in the base specification for the reasons stated below.

Two factors motivate this exclusion. First, semantic search is computationally expensive at index scale: inference on every incoming query creates a load profile incompatible with the latency and cost targets of a globally shared, freely accessible discovery endpoint. Second, short free-text queries — the form agents will most commonly submit — are insufficient input for embedding-based retrieval to reliably outperform structured keyword and capability-taxonomy filters; the signal-to-noise ratio degrades rapidly below approximately 50 tokens of context, producing ranking instability that would erode agent trust in the index.

The q parameter (free-text across name and description) combined with the structured filters defined above covers the precision requirements of well-formed agent queries. Agents requiring relevance ranking beyond what structured filters provide SHOULD perform client-side re-ranking over a filtered candidate set fetched from the index.

Agents requiring semantic selection over a filtered candidate set are encouraged to implement — or delegate to — a Semantic Routing Platform (SRP) as described in draft-cui-ai-agent-discovery-invocation ([I-D.cui-ai-agent-discovery-invocation]): an optional control-plane layer that performs semantic matching and ranking against a structured candidate pool. APIX is the natural source for that pool; an SRP queries APIX with structured filters to obtain a trusted, governed candidate set and then applies semantic ranking above that foundation. This separation keeps index infrastructure costs predictable while enabling full semantic selection capability for agents that need it.

A future premium tier of the APIX MAY introduce semantic search as a priced capability (charged per request) once a quality and cost model acceptable to the governing body and the operator community has been established. Any such capability would be offered as an optional endpoint and would not alter the behaviour of the base search endpoint defined in this specification.

1.8.2. Search Result Schema (Level 1)

Search results return lightweight summary records. These contain only the fields needed to evaluate candidates and decide which detail pages to fetch. Complex metadata (auth requirements, version history, notifications, legal, standard_warnings) is available only at Level 2 and is evaluated client-side after fetching the detail resource.

```
{
  "service_id": "svc-examplepay-v2",
  "name": "ExamplePay Payment API",
  "description": "Card and subscription payment processing",
  "api_version": "2.4.1",
  "lifecycle_stage": "stable",
  "capabilities": ["payments.card", "payments.subscription"],
  "protocol": "openapi",
  "trust": {
    "organisation_level": "O-4",
    "service_level": "S-2",
    "spec_consistency": "consistent",
    "spec_fetch_consecutive_failures": 0,
    "next_spider_run_at": "2026-04-20T14:55:00Z",
    "liveness": {
      "last_ping_at": "2026-04-20T13:55:00Z",
      "ping_interval_seconds": 3600,
      "uptime_30d_percent": 99.87,
      "consecutive_failures": 0
    }
  },
  "_links": {
    "self": {
      "href": "https://apix.example.org/services/svc-examplepay-v2"
    },
    "latest_stable": {
      "href": "https://apix.example.org/services/svc-examplepay-v2"
    }
  }
}
```


The latest_stable link points to the leaf version of the service's version chain. When latest_stable differs from self, a newer stable version exists and the agent SHOULD follow the link before proceeding.

1.8.3. Service Record Schema (Level 2)

The full Service Record is returned when a consuming agent fetches the detail resource via the self link. It is the APM plus Spider-enriched trust metadata, versioning links, and any standard_warnings.


```
{
  "service_id": "svc-examplepay-v2",
  "apm_version": "1.0",
  "name": "ExamplePay Payment API",
  "description": "Card and subscription payment processing",
  "api_version": "2.4.1",
```

```
  "lifecycle_stage": "stable",
  "supersedes": "svc-examplepay-v1",
  "superseded_by": null,
  "owner": { "...": "..." },
  "spec": {
    "type": "openapi",
    "url": "https://example.com/openapi.json",
    "version": "2.4.1"
  },
  "capabilities": ["payments.card", "payments.subscription"],
  "entry_point": "https://api.example.com/v2",
  "trust": {
    "organisation_level": "O-4",
    "organisation_verified_at": "2026-03-01T00:00:00Z",
    "organisation_verifier_id": "verifier-ch-001",
    "service_level": "S-2",
    "service_level_updated_at": "2026-04-19T08:00:00Z",
    "spec_consistency": "consistent",
    "spec_consistency_checked_at": "2026-04-20T13:55:00Z",
    "spec_fetch_consecutive_failures": 0,
    "next_spider_run_at": "2026-04-20T14:55:00Z",
    "liveness": {
      "last_ping_at": "2026-04-20T13:55:00Z",
      "ping_interval_seconds": 3600,
      "uptime_30d_percent": 99.87,
      "avg_response_ms": 142.3,
      "consecutive_failures": 0
    }
  },
  "notifications": { "...": "..." },
  "legal": { "...": "..." },
  "standard_warnings": [],
  "registered_at": "2026-01-15T00:00:00Z",
  "last_updated_at": "2026-04-20T13:00:00Z",
  "_links": {
    "self": {
      "href": "https://apix.example.org/services/svc-examplepay-v2"
    },
    "owner": {
      "href": "https://apix.example.org/organisations/org-examplepay"
    },
    "spec": { "href": "https://example.com/openapi.json" },
    "previous_version": {
      "href": "https://apix.example.org/services/svc-examplepay-v1"
    },
    "latest_stable": {
      "href": "https://apix.example.org/services/svc-examplepay-v2"
    }
```

```
    }
  }
```

1.8.4. Trust Metadata Schema

Trust metadata is always included in full Service Records (Level 2) and MUST NOT be omitted or summarised. Consuming agents rely on the full set of trust fields to evaluate their Trust Policy. Partial trust metadata MUST be represented with explicit null values, not omitted fields.

Trust metadata is included in summary form (Level 1) for server-side filter compatibility. The Level 1 trust object omits verification timestamps and verifier IDs; these are available only at Level 2.

1.8.5. Transport Encoding

The Index API is consumed by autonomous agents at machine speed. Response payloads are structured JSON with highly repetitive field names across result arrays. Transport-layer compression achieves 7085% size reduction on typical search result payloads with no information loss and no application-layer schema changes.

Compression support requirements:

The Index API MUST support the following Accept-Encoding values:

Encoding	Requirement	Notes
gzip	MUST	Universally supported baseline
br (Brotli)	SHOULD	Higher compression ratio than gzip
zstd	SHOULD	Similar ratio to Brotli; faster decompression

Table 11

The Index API MUST perform content negotiation via the Accept-Encoding request header. Responses MUST include a Content-Encoding header identifying the applied encoding. If a client sends no Accept-Encoding header, the server MAY respond uncompressed.

Consuming agents SHOULD include Accept-Encoding: zstd, br, gzip in all Index API requests.

Binary encoding (optional):

The Index API MAY additionally support CBOR ([RFC8949]) as a binary alternative to JSON. A client that prefers CBOR MUST signal this via Accept: application/cbor. The server MAY respond with Content-Type: application/cbor. CBOR responses carry identical information to JSON responses; the encoding difference is transparent to the data model.

Clients MUST NOT assume CBOR support. JSON over compressed transport is the normative interchange format.

1.9. Security Considerations

1.9.1. Abuse and Fake Listings

The mandatory Terms of Service acceptance at registration provides a first barrier against malicious actors listing fake or harmful services. For O-0 and O-1, identity verification is limited; consuming agents SHOULD NOT rely solely on index presence for trust at these levels. For O-2 and above, the formal B2B contractual relationship and progressively stronger identity and compliance verification substantially raise the cost of abuse.

Consuming agents SHOULD apply Trust Policies that exclude O-0 services for any task involving sensitive data or consequential actions.

The governing body MUST maintain an abuse reporting mechanism and MUST be able to suspend or remove a Service Record within 24 hours of confirmed abuse. Suspended service records MUST remain in the index with a status: suspended flag and MUST NOT be silently deleted, to provide transparency to agents that had cached the record.

1.9.2. Trust Level Spoofing

Organisation and Service trust levels in the Service Record are set only by the APIX itself, not by the Service Owner. APM submissions that include trust field values MUST have those values overwritten by the APIX upon processing. The Index API MUST NOT expose self-asserted trust values.

1.9.3. Transport Security Requirements

All URLs submitted as entry_point or spec.url values in an APM MUST use the https scheme. The Index MUST reject APM submissions that provide HTTP (non-TLS) values for these fields.

The {entry_point}/health endpoint MUST NOT require authentication of any kind. Requiring authentication at /health defeats liveness verification and MUST be treated as a registration defect. The Index MUST record a metadata warning if the /health endpoint returns a 401 or 407 status.

The spec.url endpoint MUST be publicly accessible without authentication. A spec.url that requires authentication cannot be verified by the Spider and MUST be treated as an S-2 failure until authentication is removed.

The Spider MUST enforce HTTPS for all outbound requests and MUST NOT follow redirects from HTTPS to HTTP.

1.9.4. Spider-Targeted Attacks

A service that knows when the Spider will visit could serve compliant responses only to Spider requests, presenting a different interface to consuming agents. Mitigations:

* Spider visit timing MUST be randomised within the liveness monitoring interval window.

* Spider User-Agent headers MUST be versioned but MUST NOT include the specific visit schedule.

* The APIX operator SHOULD perform periodic unannounced spot-checks from non-Spider infrastructure.

1.9.5. Bot Consumer Risks

The APIX provides discovery and trust metadata. It does not guarantee the safety, correctness, or availability of listed services. Consuming agents MUST NOT assume that a service listed in the APIX is safe to use without applying their own Trust Policy.

Consuming agents SHOULD treat Index API responses as untrusted input and validate the structure of Service Records before acting on them.

The Index API MUST be served exclusively over TLS. Certificate validity MUST be verified by consuming agents. Agents MUST NOT bypass TLS certificate verification when querying the Index API.

1.10. Open Questions

The following questions are unresolved and explicitly invited for community input:

1. *APM spec type extensions*: What is the formal process for registering new spec.type values? Should this be an IANA registry, or remain under the governing body governance analogous to the capability taxonomy process?

2. *Trust Policy standardisation*: Should the APIX define a standard Trust Policy expression language, or leave this entirely to consuming agents? A standard language would enable Index API server-side filtering but risks constraining agent-side policy flexibility.

3. *Regional Representative model*: How are jurisdictional boundaries defined for Regional Representatives? What happens in jurisdictions with no appointed Representative — does the central index operator serve as fallback, or is registration simply unavailable?

4. *Bot identity*: This document explicitly excludes bot identity from scope. Should a future version of the APIX include optional bot consumer registration to enable personalised discovery, rate limit management, or abuse tracking?

5. *Centralised index vs. decentralised discovery*: The APIX architecture takes a deliberate position: a single authoritative global index, governed by a neutral non-profit, with a commercial sustainability model. An alternative approach — represented by proposals such as draft-vandemeent-ains-discovery (AINS — AInternet Name Service), which uses signed, append-only replication logs with no central authority — takes the opposite position: cryptographic verifiability and censorship resistance over governed accountability.

   Arguments for the APIX model: business adoption requires a contractual counterparty; institutional co-sponsorship is only available to an accountable entity; a single entry point minimises agent-side integration complexity; the supply-side funding model creates a regional development flywheel that decentralised models cannot replicate.

   Arguments for the decentralised model: censorship resistance; no single point of failure or organisational control; cryptographic verifiability without trusting the governor.

   Community input is explicitly invited on whether the APIX and AINS-style approaches are mutually exclusive or whether a future APIX version could expose a verifiable, signed export of index records that enables third-party verification without requiring a federated registry.

6. *Geographic distribution of the index infrastructure*: This specification defines a single globally stable root entry point (https://apix.example.org/) and leaves index deployment topology to the operator. A conformant implementation MAY use CDN edge nodes, read replicas, or geographically distributed origin servers provided the HATEOAS link structure and the canonical entry-point URL remain stable. The specification intentionally does not mandate a replication or failover architecture: these are operational concerns for the governing body as index operator. A future operational profile MAY specify minimum availability, recovery time objectives, and regional failover requirements.

7. *Breach-driven trust degradation and security feed integration*: This specification defines a security_incident flag that caps a service at S-2 while active. The following questions are unresolved and invited for community input:

   * *Evidence threshold*: What constitutes sufficient evidence for the governing body to set the security_incident flag? A confirmed breach notification from the service owner, a credible third-party report, or an entry in a recognised security feed (BSI, CISA KEV, CERT-Bund, or equivalent national CERT)?

   * *Feed integration*: Should APIX consume external security feeds automatically to detect affected services, or require human review before setting a flag? Automated feed integration requires APIX to maintain knowledge of software stacks and versions per service — information not currently indexed.

   * *Aftermath window*: Should a service that has suffered a confirmed breach be required to remain at a degraded trust level for a minimum period after remediation is declared (analogous to a probationary period), or does passing a new security scan immediately restore the full S-level? If a minimum period applies, what is the appropriate duration and who determines it?

   * *Contestation process*: What is the process by which a service owner can contest a security_incident flag they believe was set in error?

   * *Real-time attack detection*: The current model is retrospective — the flag is set after a breach is confirmed. Should a future version define a real-time signal path (for example, a service owner-initiated incident declaration that

immediately and automatically caps the S-level) to allow proactive trust degradation during an active attack, before a formal breach is confirmed?

8. *Dynamic pricing negotiation*: This specification defines the "dynamic" pricing model with a pricing_endpoint that agents query to obtain the current price before invocation. The current model is accept-or-abandon: the agent queries the price, compares it against its budget ceiling, and either proceeds or abandons the invocation. A richer interaction model is conceivable — particularly for IoT services where load-based pricing varies continuously — in which an agent can submit a counter-offer (maximum acceptable price), request scheduling at a lower-load time window, or enter a brief negotiation exchange before committing to an invocation. Such a negotiation protocol would require a standardised request/response schema beyond the current pricing endpoint response, and potentially a reservation or commitment mechanism to hold a quoted price while the agent confirms with its operator. Community input is invited on whether dynamic pricing negotiation should be standardised in a future revision of this specification, and on the appropriate protocol model (synchronous counter-offer, asynchronous scheduling request, or market-based auction).

9. *Deployment region and service coverage area*: Service geography is two distinct concepts that the current APM schema does not distinguish, and both are increasingly relevant for agent-driven discovery.

   Deployment region is a data-residency and compliance concept: in which cloud regions or legal jurisdictions does the service process and store data? This matters for GDPR controller obligations, data localisation regulations, and enterprise procurement rules that prohibit cross-border data transfer. A service deployed exclusively in eu-west cannot lawfully serve queries whose data must not leave the European Economic Area.

   Service coverage area is a relevance concept: for which geographic area does the service actually provide value, independent of where its infrastructure runs? A regional organic food delivery platform running on US-hosted cloud infrastructure is still only useful to buyers and sellers located within its delivery catchment. An agent sourcing local produce for a household in Munich has no use for a service whose coverage area is the Pacific Northwest, regardless of that service’s technical availability. Coverage area is a statement about the service’s value proposition, not its infrastructure topology. For sustainability-oriented use cases — such as minimising transport

distance as a proxy for carbon footprint — the coverage area radius directly determines whether a service is a candidate for selection at all.

The two concepts can diverge: a globally deployed CDN-backed service may have a coverage area restricted to a single city; a locally hosted service may lawfully serve any jurisdiction. Conflating them into a single region field would produce incorrect filtering in both directions.

A candidate schema extension would introduce two optional fields in the APM:

json "deployment_regions": ["eu-west-1", "eu-central-1"], "coverage": { "type": "radius", "center": { "lat": 48.137, "lon": 11.576 }, "radius_km": 150 }

or alternatively a GeoJSON polygon for irregular coverage areas. Corresponding search parameters (deployment_region, near, coverage_radius_km) would allow agents to filter by both dimensions independently.

Community input is invited on: (a) the appropriate geometry primitives for coverage area (radius, bounding box, GeoJSON polygon, or NUTS/ISO 3166-2 administrative codes); (b) whether deployment region should reference cloud provider region identifiers or legal jurisdiction codes; (c) how coverage area interacts with the existing capability taxonomy for services where proximity is intrinsic to the capability (local logistics, in-person services, regional marketplaces); and (d) whether coverage area belongs in the base APM or as a profile extension.

2. References

2.1. Normative References

[APIX-CORE] Rehfeld, C., "API Index (APIX): Core Infrastructure for Autonomous Agent Service Discovery", Work in Progress, Internet-Draft, draft-rehfeld-apix-core-04, n.d., <https://datatracker.ietf.org/doc/draft-rehfeld-apix-core/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/rfc/rfc5646>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/rfc/rfc6455>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

2.2. Informative References

[APIX-IOT] Rehfeld, C., "APIX IoT Device Profile: Discovery and Presence for Connected Device Services", Work in Progress, Internet-Draft, draft-rehfeld-apix-iot-01, n.d., <https://datatracker.ietf.org/doc/draft-rehfeld-apix-iot/>.

[ASYNCAPI] AsyncAPI Initiative, "AsyncAPI Specification 3.0.0", n.d., <https://www.asyncapi.com/docs/reference/specification/v3.0.0>.

[I-D.cui-ai-agent-discovery-invocation]
Cui, Y., Chao, Y., and C. Du, "AI Agent Discovery and Invocation Protocol", Work in Progress, Internet-Draft, draft-cui-ai-agent-discovery-invocation-01, 12 February 2026, <https://datatracker.ietf.org/doc/html/draft-cui-ai-agent-discovery-invocation-01>.

[I-D.meunier-webbotauth-registry]
Guerreiro, M., Kirazci, U., and T. Meunier, "Registry and Signature Agent card for Web bot auth", Work in Progress, Internet-Draft, draft-meunier-webbotauth-registry-02, 26 May 2026, <https://datatracker.ietf.org/doc/html/draft-meunier-webbotauth-registry-02>.

[MCP] Anthropic, "Model Context Protocol", n.d., <https://modelcontextprotocol.io>.

[OPENAPI] OpenAPI Initiative, "OpenAPI Specification 3.1.0", 15 February 2021, <https://spec.openapis.org/oas/v3.1.0>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[W3C-SSE] WHATWG, "Server-Sent Events", n.d., <https://html.spec.whatwg.org/multipage/server-sent-events.html>.

Appendix A. Change Log

-00: Initial submission.

-01: Stream and intended status changed: the document moves from the Independent Submission stream (Informational) to the IETF stream (Standards Track), following IETF dispatch guidance. Capability string hierarchy formalised — "Hierarchical semantics" (is_instance_of, whole-segment containment) added to the Capability Taxonomy Registry, and "Hierarchical capability matching" (?capability= is subtree-match by default, capability_match=exact opt-out, query-time ancestor expansion) added to the search parameters. Editorial: sentence-initial capitalisation corrected.

A.1. IANA Considerations

A.1.1. Additions to the APIX Protocol Type Registry

This document requests the following additions to the Protocol Type Registry maintained by the governing body at apix.example.org/registry/protocols:

Registry value	Standard	Spider behaviour	Status
openapi	OpenAPI 3.x	Parses paths, schemas, auth requirements	active
mcp	Model Context Protocol	Parses tool definitions and capability list	active
asyncapi	AsyncAPI 2.x / 3.x	Parses channels, message schemas	active
graphql	GraphQL SDL	Introspects schema via introspection query	active

Table 12

A.1.2. Additions to the APIX Notification Channel Registry

This document requests the following additions to the Notification Channel Registry maintained by the governing body at apix.example.org/registry/notification-channels:

Registry value	Transport	Direction	Status
webhook	HTTPS POST	Server → agent	active
sse	HTTP Server-Sent Events	Server → agent	active
websocket	WebSocket (RFC 6455)	Bidirectional	active

Table 13

A.1.3. Additions to the APIX Event Type Registry

This document requests the following additions to the Event Type Registry maintained by the governing body at apix.example.org/registry/event-types:

Event type	Description	Status
spec.updated	Service spec document changed (new api_version)	active
status.changed	Service operational status changed	active
trust.changed	Organisation or service trust level changed	active

Table 14

A.1.4. Additions to the APIX Capability Taxonomy Registry

This document requests the following additions to the Capability Taxonomy Registry maintained by the governing body at apix.example.org/registry/capabilities:

Term	Description	Status
commerce	E-commerce and marketplaces	active
commerce.marketplace	Multi-vendor marketplace	active
commerce.retail	Single-vendor retail	active
payments	Payment processing	active
payments.card	Card payment processing	active
payments.crypto	Cryptocurrency payments	active
data.financial	Financial data and markets	active
data.legal	Legal documents and data	active
nlp	Natural language processing	active
nlp.translation	Language translation	active
identity	Authentication and identity	active
communication	Messaging and notifications	active
storage	File and object storage	active
compute	Code execution and computing	active
media	Image, audio, video services	active
search	Information retrieval	active

Table 15

A.2. References

A.2.1. Normative References

* [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

* [RFC5646] Phillips, A., Davis, M., "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

* [RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, December 2017.

* [RFC6455] Fette, I., Melnikov, A., "The WebSocket Protocol", RFC 6455, December 2011.

* [RFC8414] Jones, M., et al., "OAuth 2.0 Authorization Server Metadata", RFC 8414, June 2018.

* [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018.

* [RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, May 2019.

* [RFC9110] Fielding, R., et al., "HTTP Semantics", RFC 9110, June 2022.

* [APIX-CORE] Rehfeld, C., "API Index (APIX): Core Infrastructure for Autonomous Agent Service Discovery", draft-rehfeld-apix-core-04.

A.2.2. Informative References

* [RFC8949] Bormann, C., Hoffman, P., "Concise Binary Object Representation (CBOR)", RFC 8949, December 2020.

* [OPENAPI] OpenAPI Initiative, "OpenAPI Specification 3.1.0", February 2021.

* [MCP] Anthropic, "Model Context Protocol".

* [ASYNCAPI] AsyncAPI Initiative, "AsyncAPI Specification 3.0.0".

* [APIX-IOT] Rehfeld, C., "APIX IoT Device Profile: Discovery and Presence for Connected Device Services", draft-rehfeld-apix-iot-01.

* [W3C-SSE] WHATWG, "Server-Sent Events", https://html.spec.whatwg.org/multipage/server-sent-events.html.

* [I-D.meunier-webbotauth-registry] Meunier, T., et al., "Web Bot Authentication Registry".

* [I-D.cui-ai-agent-discovery-invocation] Cui, Y., et al., "AI Agent Discovery and Invocation", see Section 7.2.

A.3. Author’s Address

Carsten Rehfeld Email: carsten@botstandards.org

Author’s Address

Carsten Rehfeld
Email: carsten@botstandards.org