Independent Submission
Internet-Draft
Intended status: Informational
Expires: 9 November 2026

J. Xu
B. Li
R. Zhu
CAICT

8 May 2026

Metadata and Query Profile for Efficient Agent Discovery
draft-xu-efficient-agent-discovery-profile-00

Abstract

This document defines a metadata and query profile for efficient discovery of autonomous software agents. The profile describes how agents publish capability metadata using explicit tags, natural-language descriptions, example tasks, protocol bindings, and operational constraints. It also defines JSON metadata, discovery request, and discovery response objects that enable discovery services to combine structured filtering, semantic retrieval, and fine-grained matching while preserving an interoperable protocol surface.

The profile does not mandate a specific machine-learning model, embedding model, database, or ranking algorithm. Instead, it standardizes the metadata and result evidence needed by clients and discovery services so that different implementations can interoperate while optimizing their own latency, scalability, and ranking quality.

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 9 November 2026.

Table of Contents

1. Introduction

Agent ecosystems require a way to find suitable agents from large and dynamic candidate sets. Exact lookup by identifier is not sufficient: users and agents often express intent in natural language, while available agents expose capabilities using heterogeneous names, descriptions, tags, examples, protocols, and endpoint metadata.

Naive discovery approaches have undesirable trade-offs. Heavy language-model parsing can provide good intent understanding but may add unacceptable latency for interactive or agent-to-agent negotiation. Simple vector search over a single concatenated description can be fast but may dilute important capability details in multi-purpose agents. Keyword search over tags can be efficient but may miss semantically related agents.

This document defines an efficient agent discovery profile that separates agent metadata into complementary signals:

* explicit capability tags for fast structured filtering;

* natural-language descriptions for broad semantic matching;

* example tasks for fine-grained intent matching;

* protocol and endpoint bindings for reachability; and

* operational constraints for safe candidate selection.

The profile standardizes what agents publish and what discovery services return. Discovery services remain free to implement tag indices, dense vector indices, example-level matching, neural routers, symbolic rules, or hybrid approaches.

1.1. Problem Statement

Agent discovery has two competing requirements. It must be expressive enough to match natural-language task intent against heterogeneous agents, and it must be concrete enough that independent implementations can exchange metadata and queries without sharing the same ranking engine.

Existing mechanisms can publish web resources, service descriptors, or tool manifests, but they do not by themselves define the discovery-specific combination of capability tags, broad descriptions, example tasks, freshness, matching evidence, and ranked responses used by this profile. Without a common profile, discovery services either expose platform-specific metadata or return opaque rankings that clients cannot validate, debug, or compare.

This document therefore standardizes the metadata and query surface, not the internal retrieval implementation.

1.2. Applicability

This profile is applicable to registries, directories, marketplaces, gateways, or local discovery services that need to return candidate agents for a task intent. It is not intended to replace exact identifier resolution, endpoint security, or post-discovery invocation protocols.

1.3. Goals

This profile has the following goals:

* define an agent metadata object suitable for high-performance discovery;

* support both structured filtering and natural-language intent search;

* preserve fine-grained capability signals for multi-purpose agents;

* define discovery request and response fields that are independent of any particular ranking algorithm;

* allow discovery results to include matching evidence and confidence signals;

* support incremental metadata updates; and

* provide security and privacy guidance for discovery services.

1.4. Non-Goals

This profile does not define:

* a new transport protocol;

* a new URI scheme;

* a mandatory registry topology;

* a mandatory ranking formula;

* a mandatory embedding model, language model, or vector database;

* a reputation system; or

* an invocation protocol for executing tasks after discovery.

1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Terminology

Agent: An autonomous or semi-autonomous software system that exposes one or more capabilities and can be selected for invocation.

Agent Metadata: A structured object that describes an agent’s identity, capabilities, examples, protocol bindings, and operational constraints.

Capability Tag: A short normalized label that identifies a capability, domain, industry, task type, or other discovery-relevant property.

Context Description: A natural-language description of an agent’s purpose, scope, and supported tasks.

Discovery Request: A request sent by a client or agent to a discovery service to obtain candidate agents.

Discovery Response: A response containing ranked candidate agents and matching evidence.

Example Task: A short natural-language example of a user request, agent-to-agent request, or task pattern that the agent is intended to handle.

Matching Evidence: Data returned by a discovery service to explain why a candidate was selected, such as matched tags, matched examples, score components, or policy filters.

3. Profile Overview

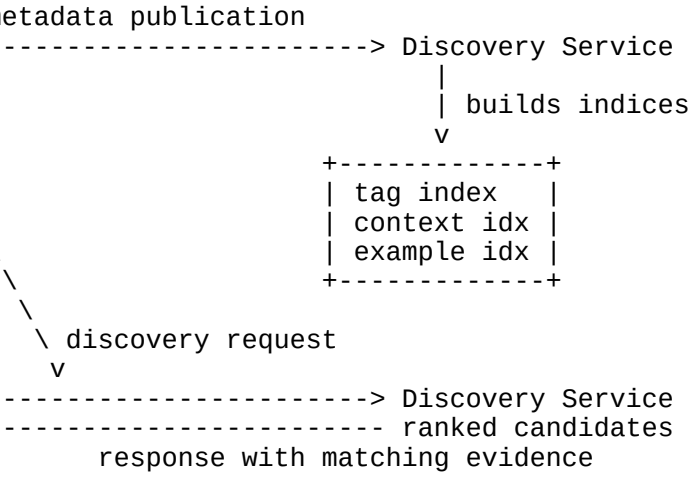
The profile has two interoperable surfaces:

* the Agent Metadata object, published by agents or registries; and

* the Discovery Request and Discovery Response objects, exchanged between clients and discovery services.

The canonical field names and examples in this profile use JSON [RFC8259]. Other serializations MAY be used by a transport or registry binding only when the binding defines an unambiguous mapping for field names, value types, arrays, objects, and unknown-field behavior.

The following figure shows a typical implementation. It is illustrative, not normative.



The discovery service MAY build separate internal indices from the same metadata. For example, it might build an inverted index for tags, an embedding index for descriptions, and an example-level index for task examples. These internal structures are not part of the wire profile.

3.1. Conformance Levels

This profile defines three conformance levels for incremental adoption.

Level	Name	Requirements
D0	Metadata	Publish Agent Metadata with id, name, description, and at least one bindings entry.
D1	Structured Discovery	Support Discovery Requests with tag and protocol filters and return ranked candidates.
D2	Evidence-Based Discovery	Return matching evidence, freshness data, and score components when requested.

Table 1

Implementations SHOULD state the highest level they support. A D2 discovery service MUST also satisfy D0 and D1. Clients SHOULD tolerate lower levels but MUST NOT assume that a D0 publisher supports query processing.

4. Agent Metadata

An Agent Metadata object describes an agent for discovery. It MUST be a single JSON object unless another serialization is explicitly selected by a binding profile that preserves the data model defined in this document.

4.1. Top-Level Fields

Field	Type	Requirement	Description
id	string	MUST	Stable identifier of the agent
name	string	MUST	Human-readable name
description	string	MUST	Natural-language context description
tags	array of string	SHOULD	Normalized capability tags
examples	array of object	SHOULD	Example tasks or request patterns
bindings	array of object	MUST	Protocol and endpoint bindings
auth	object	SHOULD	Authentication requirements
constraints	object	MAY	Region, policy, cost, rate, or data constraints
status	string	SHOULD	Operational status
expires_at	string	MAY	Time after which the record should be refreshed
version	string	SHOULD	Metadata version or capability version
updated_at	string	SHOULD	Timestamp for freshness checks
signature	object	MAY	Signature or credential binding metadata

Table 2

Field names are case-sensitive. Implementations SHOULD ignore unknown fields for forward compatibility, but MUST preserve unknown fields when acting as a transparent metadata relay.

4.2. Identifier and Name

The id field MUST be stable for the lifecycle of the agent registration. It MAY be a URI, DNS-based name, registry-local identifier, or another profile-defined identifier. This profile does not define or reserve a new URI scheme; when URIs are used, their schemes are defined outside this document. Discovery services MUST NOT assume that the human-readable name field is globally unique.

4.3. Description

The description field provides broad semantic context. It SHOULD describe:

* the agent’s primary purpose;
* supported domains;
* important input and output types;
* operational limitations; and
* safety or policy-relevant boundaries.

Descriptions SHOULD be written for discovery and interoperability, not marketing. They SHOULD avoid unsupported superlatives such as "best", "perfect", or "guaranteed".

4.4. Capability Tags

The tags field contains normalized labels. Tags SHOULD be selected from one or more declared taxonomies when possible.

Tags SHOULD be concise, lowercase, and stable. A tag SHOULD represent a capability, domain, task family, data type, industry, protocol, or language. Examples:

["translation", "finance", "invoice-processing", "json-output"]

Discovery services MAY expand tags using taxonomy relationships or synonym maps. If expansion is used, the response SHOULD distinguish directly matched tags from expanded or inferred tags.

4.5. Example Tasks

The examples field preserves fine-grained capability signals. Each example SHOULD describe a concrete task the agent can handle.

Field	Type	Requirement	Description
id	string	SHOULD	Stable example identifier
text	string	MUST	Example task or request
tags	array of string	MAY	Tags associated with the example
input_schema	object	MAY	Expected input shape
output_schema	object	MAY	Expected output shape
language	string	MAY	Language tag

Table 3

Example tasks SHOULD be specific. For a multi-purpose agent, publishers SHOULD include examples for distinct capability modes rather than averaging all capabilities into one broad description.

4.6. Protocol Bindings

The bindings field tells a client how the agent can be contacted or invoked after discovery.

Field	Type	Requirement	Description
protocol	string	MUST	Protocol identifier defined by the selected binding profile, such as https
endpoint	string	MUST	Endpoint reference or URI
media_types	array of string	MAY	Supported request or response media types
interaction_model	string	MAY	request-response, streaming, batch, or profile-defined value
priority	integer	MAY	Preference order among bindings

Table 4

Discovery clients MUST verify that a binding is acceptable under local policy before invocation.

4.7. Versioning and Extensions

Metadata publishers SHOULD include a version field and SHOULD update it whenever the meaning of capabilities, examples, or bindings changes.

Unknown top-level fields MUST be ignored by discovery services unless local policy requires strict validation. Unknown fields in nested objects SHOULD also be ignored. Implementations that relay metadata SHOULD preserve unknown fields unless doing so conflicts with privacy or policy requirements.

Private extension fields SHOULD use collision-resistant names, such as reverse-DNS names or URIs. Extensions that affect ranking, authorization, or wire compatibility SHOULD be documented in a stable specification.

4.8. Status and Freshness

The status field SHOULD use one of:

* active
* inactive
* suspended
* deprecated
* testing

The updated_at or version field SHOULD be used to reject stale metadata. Discovery services SHOULD expose when a result was indexed and whether the source metadata is older than a client-supplied freshness constraint.

If expires_at is present, discovery services SHOULD stop returning the record after that time unless local policy explicitly allows stale results. Publishers SHOULD refresh records before expiration and SHOULD use random jitter for large fleets to avoid synchronized refresh bursts.

4.9. Metadata Example

```
{
  "id": "https://agents.example.net/id/hr-core-automator",
  "name": "HR Core Automator",
  "description": "Optimizes HR workflows and onboarding checks.",
  "tags": ["hr", "workflow", "onboarding", "hcm", "api-automation"],
  "examples": [
    {
      "id": "ex-1",
      "text": "Prepare a new-employee onboarding workflow.",
      "tags": ["onboarding", "workflow"]
    },
    {
      "id": "ex-2",
      "text": "Check an employee record for missing payroll fields.",
      "tags": ["employee-record", "validation"]
    }
  ],
  "bindings": [
    {
      "protocol": "https",
      "endpoint": "https://agents.example.net/hr-core/invoke",
      "media_types": ["application/json"],
      "interaction_model": "request-response"
    }
  ],
  "status": "active",
  "version": "1.0.0",
  "updated_at": "2026-05-08T00:00:00Z"
}
```

5. Discovery Request

A Discovery Request asks a discovery service to return candidate agents. It MUST be a single JSON object unless another serialization is explicitly selected by a binding profile that preserves the data model defined in this document.

5.1. Request Fields

Field	Type	Requirement	Description
query	string	MUST	Natural-language intent or task description
required_tags	array of	MAY	Tags that candidates must satisfy

	string		
preferred_tags	array of string	MAY	Tags that increase ranking score
excluded_tags	array of string	MAY	Tags that candidates must not contain
protocols	array of string	MAY	Acceptable protocols
constraints	object	MAY	Policy, region, cost, latency, or freshness constraints
limit	integer	MAY	Maximum number of candidates
include_evidence	boolean	MAY	Whether matching evidence is requested
detail	string	MAY	minimal, summary, or full response detail
client_context	object	MAY	Client-side hints such as locale or domain

Table 5

A discovery service MUST apply required_tags and excluded_tags as hard filters when it supports the corresponding taxonomy. If it cannot apply a hard filter, it MUST report that limitation in the response.

The detail field controls metadata minimization. A value of minimal requests only identifiers, status, and enough endpoint data to continue resolution. A value of summary requests the normal candidate summary. A value of full requests complete metadata where policy permits. Discovery services MAY redact fields even when full is requested.

5.2. Request Example

```
{
  "query": "Find an agent for onboarding validation and API setup.",
  "required_tags": ["hr"],
  "preferred_tags": ["onboarding", "api-automation"],
  "protocols": ["https"],
  "constraints": {
    "max_results_age_seconds": 300,
    "region": "apac"
  },
  "limit": 10,
  "detail": "summary",
  "include_evidence": true
}
```

6. Discovery Response

A Discovery Response contains ranked candidates. It MUST be a single JSON object unless another serialization is explicitly selected by a binding profile that preserves the data model defined in this document.

6.1. Response Fields

Field	Type	Requirement	Description
request_id	string	SHOULD	Identifier for tracing the request
candidates	array of object	MUST	Ranked candidate agents
applied_filters	object	SHOULD	Filters applied by the discovery service
unsupported_filters	array of string	MAY	Filters the service could not apply
warnings	array of string	MAY	Limitations or policy warnings
generated_at	string	SHOULD	Response generation timestamp

Table 6

Each candidate object SHOULD include:

Field	Type	Requirement	Description
id	string	MUST	Agent identifier
name	string	SHOULD	Human-readable name
description	string	SHOULD	Short description or summary
bindings	array of object	SHOULD	Usable protocol bindings
score	number	SHOULD	Overall ranking score
score_components	object	MAY	Structured score details
matched_tags	array of string	MAY	Tags matched directly or by expansion
matched_examples	array of object	MAY	Examples that matched the query
credential_refs	array of string	MAY	References for verification
redacted	boolean	MAY	Whether fields were omitted by policy
status	string	SHOULD	Operational status
freshness	object	SHOULD	Index and metadata freshness

Table 7

Scores are local to the discovery service. Clients MUST NOT compare scores from different discovery services unless a common scoring profile is explicitly defined.

6.2. Score Components

When score_components is returned, the following component names are RECOMMENDED:

Component	Meaning
tag	Match between query-derived tags and agent tags
context	Semantic match against descriptions or broad context text
example	Match against one or more example tasks
protocol	Compatibility with requested protocol bindings
freshness	Penalty or bonus based on metadata age
policy	Local policy, trust, region, or entitlement signal

Table 8

Implementations MAY add deployment-specific components. Unknown components MUST be ignored by clients that do not understand them.

6.3. Response Example

```
{
  "request_id": "req-123",
  "generated_at": "2026-05-08T00:00:01Z",
  "applied_filters": {
    "required_tags": ["hr"],
    "protocols": ["https"]
  },
  "candidates": [
    {
      "id": "https://agents.example.net/id/hr-core-automator",
      "name": "HR Core Automator",
      "description": "Optimizes onboarding workflows.",
      "bindings": [
        {
          "protocol": "https",
          "endpoint": "https://agents.example.net/hr-core/invoke"
        }
      ],
      "score": 0.91,
      "score_components": {
        "tag": 0.95,
        "context": 0.83,
        "example": 0.96
      },
      "matched_tags": ["hr", "onboarding", "api-automation"],
      "matched_examples": [
        {
          "id": "ex-1",
          "score": 0.96,
          "text": "Prepare a new-employee onboarding workflow."
        }
      ],
      "status": "active",
      "freshness": {
        "metadata_updated_at": "2026-05-08T00:00:00Z",
        "indexed_at": "2026-05-08T00:00:00Z"
      }
    }
  ]
}
```


7. Discovery Processing Model

This section describes a recommended processing model. It is not mandatory, but it explains how the metadata fields in this profile support efficient discovery.

7.1. Index Construction

A discovery service can build three complementary indices:

Tag index: An inverted index from capability tag to agent identifiers. This supports fast filtering and high-recall candidate generation.

Context index: A semantic index over descriptions, generated summaries, or other broad context fields. This supports natural-language recall when user wording differs from tags.

Example index: A fine-grained index over individual example tasks. This preserves distinct capability modes for multi-purpose agents.

Discovery services MAY augment descriptions with synthetic queries or other derived text. If derived text materially affects ranking, implementations SHOULD retain provenance so that debugging and abuse review are possible.

7.2. Query Processing

A discovery service following the recommended model processes a request as follows:

1. Validate the request and apply authentication or rate limits.
2. Normalize tags and constraints against supported taxonomies.
3. Derive query features, such as predicted tags, query embeddings, or symbolic constraints.
4. Retrieve candidates from the tag index and context index.
5. Merge and deduplicate candidates.
6. Re-rank candidates using example-level, policy, freshness, and operational signals.
7. Return candidates with matching evidence and warnings.

The service SHOULD bound the number of candidates entering expensive re-ranking stages. The service SHOULD return partial results with warnings when a non-critical ranking component is unavailable.

7.3. Example-Level Matching

For multi-purpose agents, example-level matching is RECOMMENDED. The service SHOULD preserve separate example vectors or features rather than representing all examples only as one averaged vector. This allows a query to match one specific capability even when the agent has other unrelated capabilities.

A discovery service MAY compute the final score using a weighted combination of tag, context, example, freshness, and policy components. This document does not define the weights.

7.4. Privacy Defaults and Redaction

Discovery services SHOULD minimize results by default. Unless detail=full is requested and authorized, responses SHOULD omit sensitive endpoint details, private examples, internal system names, billing hints, and non-public credential references.

If fields are omitted due to policy, the candidate SHOULD include "redacted": true and the response SHOULD include a warning or policy reason when doing so does not reveal sensitive information.

8. Matching Evidence

Matching evidence helps clients decide whether a result is useful and helps operators debug search behavior.

When include_evidence is true, a discovery response SHOULD include at least one of:

* matched tags;
* matched examples;
* matched description snippets;
* score components;
* applied filters;
* rejected hard constraints; or
* freshness indicators.

Evidence MUST NOT disclose private metadata or private user queries from other clients.

9. Registration and Update Behavior

Discovery quality depends on fresh metadata. Registries or agents SHOULD submit incremental updates when:

* endpoints change;
* protocol bindings change;
* capability tags change;
* examples are added or removed;
* operational status changes; or
* credentials are revoked or renewed.

Discovery services SHOULD support sequence numbers, timestamps, entity tags, or equivalent freshness mechanisms. A stale update MUST NOT replace a newer record.

Registrations SHOULD be idempotent on (id, binding endpoint) or another profile-defined stable key. Re-sending the same key with newer freshness data SHOULD be treated as a refresh. Re-sending the same id with conflicting ownership or incompatible bindings SHOULD produce a conflict error.

10. Federation

Discovery services MAY federate results from other discovery services or registries. Federated results MUST include provenance sufficient for a client or local policy engine to determine the origin of the data.

Federated candidate metadata SHOULD include:

Field	Description
origin_service	Discovery or registry service that supplied the record
origin_authority	Authority or domain responsible for the record
origin_signature	Signature or reference covering the transferred metadata
received_at	Time the local discovery service received the record

Table 9

Federation MUST NOT remove freshness, redaction, or credential information that a client needs to make a trust decision.

11. Interoperability Considerations

This profile is intended to map cleanly to existing agent metadata formats. Where another format already defines fields such as agent cards, tool lists, OpenAPI descriptions, or protocol-specific manifests, implementers SHOULD map those fields into description, tags, examples, bindings, and constraints rather than duplicating incompatible metadata.

The following mapping is typical:

Source Concept	Profile Field
Agent card name	name
Agent card description	description
Skill or capability list	tags
Tool examples or prompts	examples
Endpoint or transport binding	bindings
Authentication metadata	auth
Rate or region limits	constraints

Table 10

12. Error Model

If this profile is carried over a request/response protocol, errors SHOULD be returned as structured objects containing:

Field	Requirement	Description
code	MUST	Stable machine-readable error code
message	SHOULD	Human-readable explanation
correlation_id	SHOULD	Identifier for tracing and support
retry_after	MAY	Suggested retry delay

Table 11

The following error codes are RECOMMENDED:

Code	Meaning
invalid_request	Required field missing or malformed
unsupported_filter	The service cannot apply a requested hard filter
unauthorized	Authentication is required or failed
forbidden	Authenticated client lacks permission
conflict	Registration conflicts with an existing record
not_found	Requested agent or record was not found
stale_metadata	Candidate or registration failed freshness checks
rate_limited	Client exceeded rate or quota limits

Table 12

HTTP bindings, if used, SHOULD map these errors to conventional HTTP status codes such as 400, 401, 403, 404, 409, 410, 429, and 503, using the semantics defined for HTTP status codes in [RFC9110].

13. Test Vectors

This section is non-normative.

The following examples can be used as minimal parser and interoperation tests.

13.1. Minimal D0 Metadata

```
{
  "id": "https://example.net/agents/minimal",
  "name": "Minimal Agent",
  "description": "Answers short factual questions.",
  "bindings": [
    {
      "protocol": "https",
      "endpoint": "https://example.net/agent/invoke"
    }
  ]
}
```

An implementation claiming D0 support MUST accept this object as valid metadata, assuming the identifier and endpoint scheme are allowed by local policy.

13.2. Minimal D1 Query

```
{
  "query": "answer a short factual question",
  "protocols": ["https"],
  "limit": 1
}
```

A D1 discovery service receiving this query SHOULD return zero or more candidates and MUST NOT require include_evidence.

13.3. Unsupported Hard Filter

```
{
  "query": "find a translation agent",
  "required_tags": ["translation"],
  "constraints": {
    "unsupported_private_filter": "example"
  }
}
```

If the service treats unsupported_private_filter as a hard filter and cannot apply it, the service MUST either fail the request or include the filter name in unsupported_filters.

14. Security Considerations

Discovery is security-sensitive. A malicious discovery result can steer a client to an attacker-controlled agent.

Metadata authenticity: Clients SHOULD verify signatures,

credentials, or registry attestations before invoking an agent returned by discovery.

Capability fraud: Agents can exaggerate descriptions, tags, or examples. Discovery services SHOULD distinguish self-asserted metadata from verified metadata.

Prompt injection in metadata: Descriptions and examples are untrusted text. Discovery services using language models MUST prevent metadata from overriding system instructions, leaking private data, or changing ranking policies outside the intended processing path.

Ranking manipulation: Attackers may stuff tags, generate misleading examples, or create many similar agents. Discovery services SHOULD apply abuse detection, duplicate detection, rate limits, and provenance checks.

Stale or revoked results: Discovery services SHOULD expose freshness data. Clients MUST check revocation and endpoint validity before invocation.

Federation trust: Federated discovery can amplify false metadata. Discovery services MUST preserve origin and signature information and SHOULD apply local trust policy before merging remote candidates into local results.

Filter bypass: A discovery service that silently ignores unsupported hard filters can return unsafe candidates. Services MUST report unsupported hard filters in unsupported_filters or fail the request.

Sensitive intent leakage: Discovery requests can reveal user goals. Discovery services MUST protect requests in transit and SHOULD minimize retention.

Result authorization: A discovery response is not an authorization decision. The target agent MUST still enforce its own access control policy.

15. Privacy Considerations

Agent metadata may reveal private business capabilities. Publishers SHOULD avoid exposing internal system names, customer identifiers, private schemas, or secrets in descriptions and examples.

Discovery services SHOULD evaluate these risks using the general privacy guidance for Internet protocols in [RFC6973].

Example tasks can reveal sensitive use cases. Publishers SHOULD provide representative but sanitized examples.

Discovery services SHOULD minimize query logging, support deletion or retention policies, and avoid using private discovery requests to train models without explicit authorization.

16. IANA Considerations

This document defines profile-local field names, string values, and error codes only. It does not create new registries, media types, URI schemes, HTTP fields, or protocol parameter values. It makes no IANA requests.

17. Implementation Status

This section is to be removed before publication as an RFC.

Prototype discovery systems can implement this profile using a tag inverted index, a dense context index over descriptions, and an example-level semantic index. Small specialized classifiers or other fast routing models may be used to predict tags, but such models are not required for interoperability.

18. Implementation Checklist

An implementation can be reviewed against the following checklist:

* supported conformance level;
* accepted metadata serialization;
* required and optional fields validated;
* unknown-field behavior documented;
* freshness and expiration checks implemented;
* hard-filter failure behavior implemented;
* score components documented;
* redaction behavior documented;
* federation provenance preserved; and
* security controls for metadata injection and ranking abuse documented.

19. References

19.1. Normative References

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[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>.

19.2. Informative References

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[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>.

Appendix A. Acknowledgements

The authors thank participants in discussions on agent discovery, metadata profiles, and efficient retrieval for large agent ecosystems.

Authors' Addresses

Jinliang Xu
CAICT
No. 52, Huayuan North Road
Beijing
Haidian District, 100191
China
Email: xujinliang@caict.ac.cn

Bingqi Li
CAICT
No. 52, Huayuan North Road
Beijing
Haidian District, 100191
China
Email: libingqi@caict.ac.cn

Runkai Zhu
CAICT
No. 52, Huayuan North Road
Beijing
Haidian District, 100191
China
Email: zhurunkai@caict.ac.cn