Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 9 November 2026

J. Jimenez
Ericsson

8 May 2026

Agent Directory
draft-jimenez-agent-directory-01

Abstract

This document defines the Agent Directory (AD), a service where agents register their identity, capabilities, and reachable endpoints and where clients discover them by capability. The AD adapts the Constrained RESTful Environments (CoRE) Resource Directory (RFC 9176) from constrained IoT devices to software agents, reusing its registration, lookup, and soft-state lifecycle model. The AD uses HTTP and JSON. MCP and A2A define how agents interact; this document defines how they are found.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-jimenez-agent-directory/.

Source for this draft and an issue tracker can be found at https://github.com/jaimejim/draft-jimenez-agent-directory.

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

The CoRE Resource Directory [RFC9176] defines a service for discovering resources hosted by other web servers, particularly in networks where direct discovery is impractical. Endpoints register links describing their resources, and clients look them up later. Registrations are soft state with a configurable lifetime. The RD provides both resource-level and endpoint-level lookup.

A similar discovery problem arises for software agents. LLM-based assistants, tool-calling agents, and multi-agent systems are deployed without fixed addresses and speak different interaction protocols (MCP [MCP], A2A [A2A], gRPC). Direct enumeration of candidate endpoints does not scale beyond a single administrative domain.

Individual agents may publish their own metadata at well-known endpoints (such as the A2A Agent Card at /.well-known/agent-card.json). However, per-agent metadata requires knowing the agent's URL before you can discover what it offers. The AD aggregates per-agent metadata into a queryable service. Clients discover agents by capability without prior knowledge of their addresses.

Like the CoRE RD, the AD has a small footprint: it can run as a cloud service, as a sidecar alongside an agent orchestrator, on an edge gateway, or on a constrained device. Deployment profiles range from cloud services to constrained devices colocated with sensors and actuators.

The lookup interface is a small set of HTTP GET requests with a fixed set of query parameters. Clients include both conventional applications, which use structured filters, and LLM-based clients, which can additionally match on the natural-language descriptions and tags carried in registrations. The interface is small enough to be described as a tool in an MCP server or function-calling schema.

This document specifies the Agent Directory (AD). The core mapping to CoRE RD concepts is:

CoRE RD Concept	AD Equivalent
Endpoint (EP)	Agent
Resource link	Capability / tool description
Registration resource	Agent registration resource
Endpoint lookup	Lookup (GET on the lookup endpoint)
Resource lookup	Capabilities embedded in the agent registration
Lifetime (lt)	Registration lifetime

Table 1

Agents register with the AD by sending a POST request with a JSON document describing their capabilities. Registrations are soft state and expire unless refreshed.

1.1. Terminology

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 [BCP14] (RFC2119) (RFC8174) when, and only when, they appear in all capitals, as shown here.

This specification makes use of the following terminology:

Agent:
: An autonomous or semi-autonomous software entity that can initiate and receive interactions, expose capabilities (tools, skills, APIs), and optionally collaborate with other agents.

Agent Directory (AD):
: An HTTP service that stores agent registrations and provides discovery and lookup interfaces.

Capability:
: A discrete function, tool, or skill that an agent exposes. Capabilities are described as entries within an agent’s registration.

Registration:
: The act of an agent (or a commissioning tool acting on its behalf) submitting its identity, capabilities, and endpoint information to the AD.

Registration Resource:
: The resource stored at the AD as a result of a successful registration. Identified by the URI returned in the Location header of the creation response.

Interaction Protocol:
: The application-level protocol through which an agent can be reached for task execution, such as MCP [MCP], A2A [A2A], or gRPC. HTTP is the underlying transport for most interaction protocols but is not itself an interaction protocol in this context.

2. Architecture

Figure 1 shows the AD architecture.

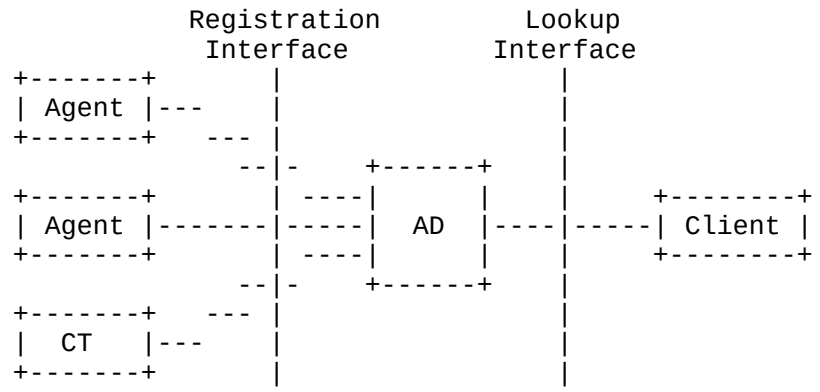

Figure 1: Agent Directory Architecture

Agents register their capabilities and endpoints with the AD. Clients (which may themselves be agents) query the AD's lookup interface to discover agents matching desired criteria. A Commissioning Tool (CT) may register agents on their behalf, for example when a platform operator manages a fleet of agents. How a CT proves authority to act on behalf of an agent is deployment-specific and out of scope for this document.

2.1. Principles

The AD operates on the following principles:

* The AD stores information that could otherwise only be obtained by directly querying each agent's metadata endpoint.

* Registrations are soft state. The registering agent (or its CT) MUST periodically refresh each registration before its lifetime expires.

* The AD MUST NOT permit modification of a registration by any entity other than the original registrant or an authorized CT.

* The AD does not proxy agent interactions; it only provides discovery.

2.2. Content Model

An AD contains zero or more agent registrations. Each registration has:

* An agent name ("agent", a Unicode string) unique within the AD instance.

* A registration base URI ("base") [RFC3986], typically the agent's reachable address.

* A lifetime ("lt") in seconds.

* A registration resource location within the AD ("href").

* Optionally, a version string ("version").

* Optionally, a vendor string ("vendor").

* Zero or more capabilities, each described as a JSON object with at minimum a name and a type.

2.3. Federation

In deployments that span multiple administrative domains or geographic regions, multiple AD instances may operate independently. Federation allows these instances to share registration information so that a client querying one AD can discover agents registered at another.

This document does not specify a federation protocol. Possible approaches include periodic synchronization between peered AD instances, a publish/subscribe notification mechanism, and a hierarchical model where a root AD aggregates entries from subordinate instances. The soft-state model bounds staleness: federated entries carry the original lifetime and expire without explicit deletion if synchronization lapses.

3. AD Discovery

3.1. Well-Known URI

An AD MUST be discoverable at the well-known URI:

/.well-known/ad

A GET request to this URI returns a JSON document describing the AD's interfaces. The response object contains the following fields:

registration:
: (string, REQUIRED) Path to the registration endpoint.

lookup:
: (string, REQUIRED) URI Template [RFC6570] for the lookup endpoint. The template variables indicate the supported query parameters.

max_count:
(integer, REQUIRED) Maximum value the AD accepts for the count pagination parameter.

Example:

```
GET /.well-known/ad HTTP/1.1
Host: directory.example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "registration": "/ad/r",
  "lookup": "/ad/l{?agent,protocol,cap_name,cap_type,tag,page,count}",
  "max_count": 100
}
```

3.2. DNS-SD

An AD MAY advertise itself on a local network via DNS-SD [RFC6763] using the service name _ad._tcp. This provides zero-configuration discovery of a local AD, which is useful on networks where agents are colocated with sensors and actuators.

Several DNS-based mechanisms have been proposed in the DAWN working group for wide-area agent discovery. DNS-AID [I-D.mozleywilliams-dnsop-dnsaid] publishes agent metadata under _agents subdomains using SVCB records. DN-ANR [I-D.cui-dns-native-agent-naming-resolution] defines a resolution layer using FQDNs and SVCB/HTTPS records. These mechanisms handle naming and resolution: mapping an agent name to a network location. The AD handles the layer above: finding agents by capability without prior knowledge of their names. DNS resolves the AD's hostname; the AD carries the capability metadata that DNS cannot efficiently carry.

4. Registration

4.1. Registration Request

An agent registers by sending a POST request to the AD's registration endpoint. All HTTP interactions with the AD follow the semantics defined in [RFC9110]. The request body is a JSON object containing the agent's metadata and capabilities.

```
POST /ad/r?agent=summarizer-v2 HTTP/1.1
Host: directory.example.com
Content-Type: application/json

{
  "base": "https://agents.example.com/summarizer-v2",
  "description": "Summarizes documents and extracts named entities",
  "protocols": ["a2a"],
  "capabilities": [
    {
      "name": "summarize",
      "type": "tool",
      "description": "Summarize a document or text passage",
      "input_schema": {
        "type": "object",
        "properties": {
          "text": {"type": "string"},
          "max_length": {"type": "integer"}
        },
        "required": ["text"]
      }
    },
    {
      "name": "extract_entities",
      "type": "tool",
      "description": "Extract named entities from text"
    }
  ],
  "version": "2.1.0",
  "vendor": "Example Corp",
  "identity": "https://registry.example.com/agents/summarizer-v2",
  "identity_type": "aip"
}

HTTP/1.1 201 Created
Location: /ad/r/4521
```

The query parameters are:

agent:
: REQUIRED. Agent name. MUST be unique within the AD instance. SHOULD be short and meaningful; the AD MAY enforce a maximum length.

lt:

Lifetime in seconds (OPTIONAL). Default: 86400 (24 hours). Range: 60 to 4294967295 (2^32-1, matching [RFC9176]). The AD SHOULD enforce a maximum lifetime; a recommended default maximum is 604800 seconds (7 days).

The body JSON object contains:

base:
(string, REQUIRED) The agent's reachable base URI [RFC3986].

description:
(string, OPTIONAL) A short human-readable description of what the agent does. SHOULD be one or two sentences. Returned in lookup responses to help clients evaluate agents without additional round-trips.

protocols:
(array of strings, OPTIONAL) Interaction protocols supported. The following values are defined by this document: "mcp" (Model Context Protocol), "a2a" (Agent-to-Agent Protocol), "grpc" (gRPC). Implementations SHOULD use lowercase short names and MAY append a version (e.g., "mcp/1.0"). A future version of this document may define an IANA registry for protocol identifiers.

capabilities:
(array of objects, OPTIONAL) The agent's capabilities. Each object MUST have "name" (string) and "type" (string) fields. The "type" field indicates the kind of capability (see Section 4.1.1). Additional fields such as "description" (string), "tags" (array of strings), "input_schema" (object), and "output_schema" (object) are OPTIONAL. When present, input_schema and output_schema SHOULD be JSON Schema objects. Capability names MUST be unique within a registration.

version:
(string, OPTIONAL) The agent's version identifier.

vendor:
(string, OPTIONAL) The organization or individual that provides the agent.

identity:
(string, OPTIONAL) A URI pointing to the agent's identity metadata document. This document describes the agent's verified identity, trust posture, owner binding, or credentials in a structured format. The AD stores and returns this URI without interpreting its contents.

identity_type:
: (string, OPTIONAL) The schema or format of the identity document referenced by identity. Values include "aip" (Agent Identity Profile), "oidc" (OpenID Connect Discovery), "wimse" (WIMSE workload identity), and "did" (Decentralized Identifier Document). Clients that recognize the type can dereference and validate the identity document; clients that do not recognize the type SHOULD ignore both fields.

When present, the identity field lets a client verify an agent beyond trusting the AD. For example, a client discovering a financial-analysis agent can fetch its AIP document to check owner binding, attestation state, and credential lifetime before invoking the agent. The AD stores the URI and does not interpret it; trust decisions remain the client's. Clients MUST NOT treat the presence of an identity field as proof of identity without verifying the referenced document.

4.1.1. Capability Types

The "type" field in a capability object indicates the kind of function the agent exposes. The following initial values are defined:

tool:
: A discrete, stateless function that accepts structured input and returns structured output.

skill:
: A higher-level, potentially multi-step behavior that may involve internal reasoning, planning, or coordination. Unlike a tool, a skill may maintain conversational state.

resource:
: A data source or content provider that the agent can expose for reading.

prompt:
: A reusable prompt template that the agent can execute with supplied parameters.

This list is extensible. Implementations MAY use additional type values. The remaining types mirror those defined by MCP [MCP].

4.2. Registration Semantics

Registration is idempotent on the agent name; a POST with the same agent name acts as an upsert. This follows the RFC 9176 pattern of POST-to-collection for registration rather than PUT to a canonical agent URL. The AD stores the agent value from the query parameter as part of the registration resource and includes it in all response representations.

* If no registration exists for the agent name, a new registration resource is created. The AD returns 201 (Created) with a Location header pointing to the registration resource.

* If a registration already exists for the agent name and the request comes from the same authenticated entity, the registration is replaced with the new content. The AD returns 200 (OK) with the Location header of the existing registration resource.

* If a registration already exists for the agent name but is owned by a different authenticated entity, the AD returns 409 (Conflict).

The response body for both 201 and 200 is empty. Clients that need the full representation SHOULD send a GET request to the URI in the Location header. The AD MAY grant a lifetime shorter than requested; the granted lifetime MUST be indicated in the response via a lt field in a JSON body or via the Location header’s associated resource.

4.3. Reading a Registration

An agent or client retrieves a single registration by sending a GET request to the registration resource.

```
GET /ad/r/4521 HTTP/1.1
Host: directory.example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "agent": "summarizer-v2",
  "base": "https://agents.example.com/summarizer-v2",
  "protocols": ["a2a"],
  "capabilities": [
    {
      "name": "summarize",
      "type": "tool",
      "description": "Summarize a document or text passage",
      "input_schema": {
        "type": "object",
        "properties": {
          "text": {"type": "string"},
          "max_length": {"type": "integer"}
        },
        "required": ["text"]
      }
    },
    {
      "name": "extract_entities",
      "type": "tool",
      "description": "Extract named entities from text"
    }
  ],
  "version": "2.1.0",
  "vendor": "Example Corp",
  "href": "/ad/r/4521"
}
```

The response includes the full registration content, including capability details (description, schemas) that are omitted from lookup results (Section 5.2). If the registration resource does not exist, the AD MUST return 404 (Not Found).

4.4. Registration Update

An agent refreshes or updates its registration by sending a POST request to its registration resource (the URI returned in the Location header at creation time).

```
POST /ad/r/4521 HTTP/1.1
Host: directory.example.com

HTTP/1.1 204 No Content
```

An empty POST resets the lifetime. The agent MAY include query parameters (lt) to update those values, or a JSON body to replace the capabilities. If the registration has already expired, the AD MUST return 404 (Not Found); the agent must re-register via the registration endpoint.

An agent may update the lifetime via the lt query parameter:

```
POST /ad/r/4521?lt=3600 HTTP/1.1
Host: directory.example.com

HTTP/1.1 204 No Content
```

If the registration has already expired, the update fails:

```
POST /ad/r/4521 HTTP/1.1
Host: directory.example.com

HTTP/1.1 404 Not Found
Content-Type: application/problem+json

{
  "type": "https://ad.example.com/errors/registration-expired",
  "title": "Registration has expired",
  "status": 404
}
```

4.5. Registration Removal

An agent removes its registration by sending a DELETE request to its registration resource.

```
DELETE /ad/r/4521 HTTP/1.1
Host: directory.example.com

HTTP/1.1 204 No Content
```

The AD MUST automatically remove a registration when its lifetime elapses without a refresh.

5. Lookup

The AD provides a lookup endpoint for discovering registered agents. The lookup endpoint URI is obtained from the well-known response (Section 3). All query parameters act as conjunctive filters: only agents matching all specified criteria are returned. A request with no filters returns all agents visible to the requesting client.

```
GET /ad/l?cap_name=purge* HTTP/1.1
Host: directory.example.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

{
  "agents": [
    {
      "agent": "cdn-cache-manager",
      "base": "https://agents.example.com/cdn-cache-manager",
      "description": "Manages CDN cache invalidation and prefetch policies",
      "protocols": ["a2a"],
      "capabilities": [
        {"name": "purge_by_tag", "type": "tool"},
        {"name": "prefetch_origins", "type": "tool"}
      ],
      "href": "/ad/r/4521"
    }
  ]
}
```

5.1. Query Parameters

All filters use exact match on the registered value, with one exception: agent and cap_name support a single trailing * as a prefix-match operator (e.g., cap_name=purge* matches purge_by_tag). The * character MUST NOT appear anywhere else in these values, and agent and capability names MUST NOT contain a literal *. The AD MUST reject registrations whose agent or capability names contain * with 400 (Bad Request). No other glob or regular-expression syntax is supported.

agent:
: Filter by agent name. Trailing * for prefix match; exact match otherwise.

protocol:

Filter by interaction protocol. Matches when the given value appears in the agent's protocols array.

cap_name:
Filter by capability name. Trailing * for prefix match.

cap_type:
Filter by capability type (e.g., "tool", "skill").

tag:
Filter by capability tag. Matches when the given value appears in any capability's tags array.

page:
Page number (zero-based). Default: 0.

count:
Results per page. The AD MUST clamp values exceeding the max_count reported in the well-known response (Section 3) to that maximum. Default: the AD's max_count.

When multiple cap_* or tag filters are present, the AD MUST match agents that have at least one _single_ capability satisfying all cap_* and tag filters jointly. For example, cap_type=tool&tag=search matches agents with a capability that is both typed tool and tagged search, not agents that happen to have a tool capability and, separately, a capability with a search tag.

The AD MUST ignore unknown query parameters rather than returning an error.

5.2. Lookup Response

Each object in the agents array contains:

agent:
The registered agent name.

base:
The agent's base URI.

description:
The agent's description, if registered.

protocols:
Interaction protocols the agent supports.

capabilities:

An array of capability summary objects, each with "name" and "type". Full details (descriptions, schemas, tags) are omitted to keep responses compact.

href:
The registration resource URI. Clients retrieve the full registration by sending GET to this URI (Section 4.3).

This two-step pattern mirrors the RD's separation between endpoint lookup and resource lookup [RFC9176]: the lookup returns compact summaries with pointers; the client dereferences the pointer for full details.

5.3. Pagination

When more results exist beyond the current page, the AD includes a Link header [RFC8288] with rel="next":

```
Link: </ad/l?cap_name=purge*&page=1>; rel="next"
```

An empty agents array with no rel="next" link indicates no results or end of results.

6. Error Responses

When the AD cannot fulfill a request, it MUST return an appropriate HTTP status code and SHOULD include a problem details object as defined in [RFC9457]. The type URIs shown below are illustrative; this document does not register them.

```
HTTP/1.1 409 Conflict
Content-Type: application/problem+json

{
  "type": "https://ad.example.com/errors/agent-name-taken",
  "title": "Agent name already registered",
  "detail": "Agent name already registered by another entity.",
  "status": 409
}
```

The following error conditions are defined:

400 Bad Request:
The request body is malformed or missing required fields.

401 Unauthorized:
The request lacks valid authentication credentials.

403 Forbidden:
The authenticated entity is not authorized for the requested operation.

404 Not Found:
The referenced registration resource does not exist.

409 Conflict:
The agent name is already registered by a different entity.

429 Too Many Requests:
The client has exceeded the rate limit.

7. Security Policies

The security model is derived from Section 7 of [RFC9176] and adapted for agents. Policies for capability attestation and cross-domain trust are out of scope for this document.

7.1. Agent Name and Registration Ownership

The AD MUST ensure that a registering agent is authorized to use the given agent name. The default policy is "First Come First Remembered" as described in Section 7.5 of [RFC9176]: accept registrations for any agent name not currently active, and only accept updates from the same authenticated entity. Whether the AD should return 409 (Conflict) or 403 (Forbidden) when a different entity tries to claim an existing name is an open design question; this document uses 409 to signal a naming conflict rather than an authorization failure.

7.2. Capability Confidentiality

The AD MUST enforce access control on lookup results when the operator policy designates some registrations as restricted. Not every client is entitled to see every registration.

7.3. Capability Vetting

The AD SHOULD restrict which capability types and names an agent may claim, based on operator policy. For example, registration of an agent as a "financial-analysis" tool can be limited to agents presenting the corresponding credentials. The policy mechanism is deployment-specific and out of scope for this document.

8. Security Considerations

8.1. Transport Security

All communication with the AD MUST be protected using TLS.

8.2. Authentication

Agents MUST authenticate when registering. The AD MUST support OAuth 2.0 bearer tokens [RFC6750] or mutual TLS (mTLS). API keys are acceptable for development but do not satisfy the authentication requirement for production use.

Registration requires a verified agent identity: the AD must know who is registering before it can enforce ownership of the registration resource. Bearer tokens prove authorization but not workload identity. For production deployments, the AD SHOULD accept workload identity credentials as defined by the WIMSE working group [I-D.ietf-wimse-arch]. The WIMSE architecture treats agents as workloads and defines a wimse: URI scheme [I-D.ietf-wimse-identifier] that can serve as the agent's authenticated identity at registration time. This provides a cryptographically verifiable binding between the registrant and the registration.

If an agent's credentials are compromised, the attacker can modify the registration (including the base URI) until the credentials are revoked. Deployments SHOULD keep credential lifetimes comparable to registration lifetimes, and SHOULD log registration changes for audit.

8.3. Authorization

The AD MUST enforce that only the original registrant (or an authorized CT) can modify or delete a registration. The AD SHOULD implement rate limiting on both registration and lookup endpoints, and MUST enforce limits on registration size and number of capabilities per agent.

8.4. Privacy

Agent metadata can reveal organizational structure: a lookup response listing all agents discloses which capabilities an organization has deployed. The AD SHOULD return only those registrations that the requester is authorized to see. The interaction between scoped responses and federation is out of scope for this document.

8.5. Adversarial Metadata

Capability descriptions and tags are free-form text that LLM-based clients may use as input to their decisions. A malicious registrant can craft descriptions that bias those decisions toward invoking the wrong agent. A malicious registrant can also set base to a victim's endpoint, redirecting traffic intended for the registered agent.

Clients SHOULD treat registration metadata as untrusted input. Clients MAY verify a registration by fetching the agent's own metadata at base and comparing it with the AD response before invocation.

9. IANA Considerations

9.1. Well-Known URI Registration

This document requests registration of the following well-known URI [RFC8615]:

URI suffix:
ad

Change controller:
IETF

Specification document:
[RFC-XXXX]

9.2. Media Type

This document uses application/json for all request and response bodies and application/problem+json [RFC9457] for error responses.

10. References

10.1. Normative References

[BCP14] Best Current Practice 14,
<https://www.rfc-editor.org/info/bcp14>.
At the time of writing, this BCP comprises the following:

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/info/rfc2119>.

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/info/rfc8174>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

[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>.

[RFC9176] Ams端ss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/rfc/rfc9457>.

10.2. Informative References

[A2A] "Agent-to-Agent Protocol", 2025, <https://a2aprotocol.ai/>.

[AgentRegistry] "agentregistry", 2026, <https://aregistry.ai/>.

[ANP-Discovery]
"ANP Agent Discovery Protocol Specification", 2024, <https://agentnetworkprotocol.com/en/specs/08-anp-agent-discovery-protocol-specification/>.

[Azure-API-Center]
"Agent registry in Azure API Center", 2026, <https://learn.microsoft.com/en-us/azure/api-center/agent-to-agent-overview>.

[ERC-8004] "ERC-8004: Trustless Agents", 2026, <https://eips.ethereum.org/EIPS/eip-8004>.

[I-D.cui-dns-native-agent-naming-resolution]
Cui, Y., "DNS-Native AI Agent Naming and Resolution", Work in Progress, Internet-Draft, draft-cui-dns-native-agent-naming-resolution-01, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-cui-dns-native-agent-naming-resolution-01>.

[I-D.hood-independent-agtp]
Hood, C., "Agent Transfer Protocol (AGTP)", Work in Progress, Internet-Draft, draft-hood-independent-agtp-06, 27 April 2026, <https://datatracker.ietf.org/doc/html/draft-hood-independent-agtp-06>.

[I-D.ietf-wimse-arch]
Salowey, J. A., Rosomakho, Y., and H. Tschofenig, "Workload Identity in a Multi System Environment (WIMSE) Architecture", Work in Progress, Internet-Draft, draft-ietf-wimse-arch-07, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-07>.

[I-D.ietf-wimse-identifier]
Rosomakho, Y. and J. A. Salowey, "Workload Identifier", Work in Progress, Internet-Draft, draft-ietf-wimse-identifier-02, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-identifier-02>.

[I-D.mozleywilliams-dnsop-dnsaid]
Mozley, J., Williams, N., Sarikaya, B., and R. Schott, "DNS for AI Discovery", Work in Progress, Internet-Draft, draft-mozleywilliams-dnsop-dnsaid-01, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-mozleywilliams-dnsop-dnsaid-01>.

[I-D.mp-agntcy-ads]
Muscariello, L. and R. Polic, "Agent Directory Service", Work in Progress, Internet-Draft, draft-mp-agntcy-ads-01, 24 February 2026, <https://datatracker.ietf.org/doc/html/draft-mp-agntcy-ads-01>.

[I-D.narajala-ans]
Huang, K., Narajala, V. S., Habler, I., and A. Sheriff, "Agent Name Service (ANS): A Universal Directory for Secure AI Agent Discovery and Interoperability", Work in Progress, Internet-Draft, draft-narajala-ans-00, 16 May 2025, <https://datatracker.ietf.org/doc/html/draft-narajala-ans-00>.

[I-D.pioli-agent-discovery]
Pioli, R., "Agent Registration and Discovery Protocol (ARDP)", Work in Progress, Internet-Draft, draft-pioli-agent-discovery-01, 24 February 2026, <https://datatracker.ietf.org/doc/html/draft-pioli-agent-discovery-01>.

[I-D.zyyhl-agent-networks-framework]
Zhouye, Yao, K., Yu, M., Han, M., and C. Li, "Framework for AI Agent Networks", Work in Progress, Internet-Draft, draft-zyyhl-agent-networks-framework-01, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-zyyhl-agent-networks-framework-01>.

[MCP] "Model Context Protocol Specification", 2025, <https://spec.modelcontextprotocol.io/>.

[MCP-Registry]
"Model Context Protocol Registry", 2025, <https://github.com/modelcontextprotocol/registry>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/rfc6690>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/rfc/rfc6763>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

Appendix A. Relationship to Other Work

A.1. CoRE Resource Directory (RFC 9176)

The AD is a direct adaptation of the CoRE RD. The registration, update, removal, and lookup interfaces follow the same patterns. The differences reflect the shift from CoAP and CoRE Link Format [RFC6690] to HTTP and JSON: structured capability objects replace flat link attributes, and interaction protocol is advertised as a first-class field.

A.2. Per-Agent Metadata Endpoints

A2A [A2A] serves an Agent Card at /.well-known/agent-card.json; MCP [MCP] exchanges server metadata during initialization. Both describe a single agent and both require the client to already know the agent’s URL.

The AD complements these by aggregating metadata from many agents into a queryable registry. Clients perform capability-based search without prior knowledge of agent addresses. Once the AD returns a matching agent’s base URI, the client MAY fetch the agent’s own metadata endpoint for protocol-specific details before initiating interaction.

A.3. MCP Registry

The Model Context Protocol project maintains a public registry [MCP-Registry] where MCP servers are catalogued for discovery by MCP-compatible clients. The registry and the AD address overlapping concerns from different ends: the MCP Registry is scoped to MCP servers and curated by a single project; the AD is protocol-agnostic (MCP, A2A, gRPC, and others are all describable) and is specified here as an open interface that any operator can implement. An agent registered in the MCP Registry can also appear in an AD instance without conflict; the two are complementary rather than substitutable.

A.4. Agent Transfer Protocol (AGTP)

[I-D.hood-independent-agtp] proposes an agent-native transport protocol over TCP/TLS or QUIC, with agent identity, session semantics, and authorization expressed at the wire level rather than layered on HTTP. The AD takes the opposite approach, reusing HTTP and JSON as a substrate. The two are not mutually exclusive: an AD instance could register agents reachable over AGTP in the same way it registers agents reachable over MCP, A2A, or gRPC, by carrying the relevant protocol identifier in the protocols field.

A.5. Agent Directory Service (ADS)

[I-D.mp-agntcy-ads] specifies a system with goals similar to the AD.

ADS takes a fully decentralized approach: content-addressed identifiers (CIDs) for records, a libp2p Kad-DHT for routing, OCI-compliant storage (ORAS/zot) for retrieval, and gRPC for the client interface. It defines OASF, a hierarchical skill taxonomy with 15 top-level categories and 100+ specific skills, used for capability-based routing through the DHT. This provides federation and cryptographic integrity at the cost of requiring DHT peers, OCI registries, and gRPC infrastructure.

The AD makes the opposite trade-off: a single HTTP/JSON service with no additional infrastructure. Federation is not built in (see Section 2.3) and there is no content-addressed integrity model. ADS targets decentralized deployments where no single operator is trusted; the AD targets deployments where a managed directory is acceptable. Interoperability between the two (an AD instance indexing ADS records, or vice versa) is not specified in this document.

A.6. Agent Registration and Discovery Protocol (ARDP)

[I-D.pioli-agent-discovery] defines a similar registration and discovery service with a custom agent: identifier scheme. The AD differs by reusing standard HTTP URIs for agent identity and by grounding the design in the RD model.

A.7. DNS-Based Agent Discovery

[I-D.narajala-ans] (ANS) proposes DNS-based agent discovery with PKI certificates. The AD operates at the application layer and does not require DNS infrastructure changes.

DN-ANR [I-D.cui-dns-native-agent-naming-resolution] defines a thin DNS resolution layer using FQDNs and SVCB records, explicitly leaving capability-based discovery to an upper layer. The AD can serve as that upper layer. DNS-AID [I-D.mozleywilliams-dnsop-dnsaid] places agent metadata in DNS zones for intra-organization discovery; the AD provides cross-organization lookup. The two approaches are complementary.

A.8. Platform-Specific and On-Chain Registries

General-purpose service registries (Consul, etcd, Kubernetes service discovery) solve service location but do not model agent capabilities, interaction protocols, or soft-state registration semantics. The AD addresses the agent-specific aspects that these systems lack.

Azure API Center [Azure-API-Center] provides a centralized agent registry within the Azure ecosystem. The agentregistry project [AgentRegistry] offers a SaaS registry with a CLI. ERC-8004 [ERC-8004] defines on-chain agent registries on Ethereum.

These systems are each tied to a single provider: a cloud vendor, a SaaS operator, or a blockchain. The AD is specified by this document and can be operated by anyone; clients authenticate to the instance they use.

A.9. Agent Network Protocol (ANP)

ANP's Agent Discovery Protocol [ANP-Discovery] defines active discovery via /.well-known/agent-descriptions and passive discovery where agents register with a search service. The AD's registration model corresponds to ANP's passive mode but specifies the registration API concretely. ANP uses JSON-LD and DIDs; the AD uses plain JSON and HTTP URIs. The broader ANP framework is described in [I-D.zyyhl-agent-networks-framework].

A.10. Agent Identity Profiles

The AD's identity and identity_type fields provide an extension point for linking registrations to external identity metadata. The Agent Identity Profile (AIP) is one such format, defining a JSON document that describes an agent's owner binding, capabilities, attestation state, trust posture, and credential lifecycle. Other formats include OpenID Connect Discovery documents and WIMSE workload identity endpoints. The AD does not mandate any particular identity schema; it provides the pointer so that clients can verify agents through whatever trust framework their deployment requires.

Appendix B. Examples

B.1. Discovery Flow

Discovering an agent through the AD involves four steps: locating the directory, discovering its interfaces, querying for a matching capability, and contacting the agent directly.

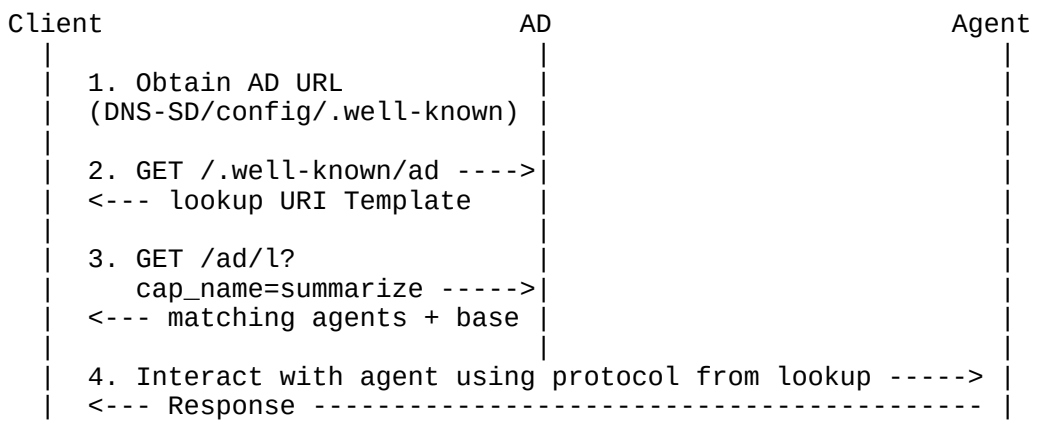

Figure 2: Discovery Flow

The steps in detail:

Step 1: Obtain the AD URL.

The client already knows the URL of an Agent Directory. This may have been learned through a DNS-SD browse for _ad._tcp, a .well-known/ad query to a known host, or simply through static configuration.

https://ad.example.com

Step 2: Discover the AD’s lookup interface.

The client queries the well-known endpoint to learn the available interfaces.

```
GET https://ad.example.com/.well-known/ad HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json

{
  "registration": "/ad/r",
  "lookup": "/ad/l{?agent,protocol,cap_name,cap_type,tag,page,count}",
  "max_count": 100
}
```

In some deployments the lookup interface path is already known or follows a convention, in which case this step can be skipped.

Step 3: Search for an agent with the desired capability.

The client expands the lookup URI Template with cap_name=summarize.

```
GET https://ad.example.com/ad/l?cap_name=summarize HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json

{
  "agents": [
    {
      "agent": "summarizer-v2",
      "base": "https://agents.example.com/summarizer-v2",
      "description": "Summarizes documents and extracts named entities",
      "protocols": ["a2a"],
      "capabilities": [
        {"name": "summarize", "type": "tool"},
        {"name": "extract_entities", "type": "tool"}
      ],
      "href": "/ad/r/4521"
    }
  ]
}
```

The response includes the agent's base URI and supported protocols, giving the client enough information to contact the agent directly.

Step 4: Interact with the agent.

The lookup response indicated that summarizer-v2 is reachable over A2A. The client fetches the agent's A2A Agent Card to obtain the full task interface before invocation.

```
GET https://agents.example.com/summarizer-v2/.well-known/agent-card.json
HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json

{ ... A2A Agent Card ... }
```

The subsequent task invocation is defined by the interaction protocol (A2A in this case, MCP or gRPC in others) and is out of scope for this document.

B.2. Enterprise Agent Portfolio

A Publisher at example.com operates three agents behind a single AD.

Step 1: Agents register with the AD.

```
POST /ad/r?agent=ticket-classifier HTTP/1.1
Host: ad.example.com
Content-Type: application/json

{
  "base": "https://agents.example.com/ticket-classifier",
  "description": "Classifies incoming support tickets.",
  "protocols": ["mcp"],
  "capabilities": [
    {"name": "classify_ticket", "type": "tool"},
    {"name": "suggest_priority", "type": "tool"}
  ],
  "vendor": "Example Corp"
}

HTTP/1.1 201 Created
Location: /ad/r/201

POST /ad/r?agent=knowledge-lookup HTTP/1.1
Host: ad.example.com
Content-Type: application/json

{
  "base": "https://agents.example.com/kb",
  "description": "Searches internal knowledge base.",
  "protocols": ["mcp"],
  "capabilities": [
    {"name": "search_kb", "type": "tool", "tags": ["nlp", "search"]}
  ],
  "vendor": "Example Corp"
}

HTTP/1.1 201 Created
Location: /ad/r/202

POST /ad/r?agent=order-router HTTP/1.1
Host: ad.example.com
Content-Type: application/json

{
  "base": "https://agents.example.com/order-router",
  "description": "Routes orders to fulfillment systems.",
  "protocols": ["a2a"],
  "capabilities": [
    {"name": "route_order", "type": "tool"}
  ],
```

```
  "vendor": "Example Corp"
}

HTTP/1.1 201 Created
Location: /ad/r/203

Step 2: A client queries for MCP agents.

GET /ad/l?protocol=mcp HTTP/1.1
Host: ad.example.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

{
  "agents": [
    {
      "agent": "ticket-classifier",
      "base": "https://agents.example.com/ticket-classifier",
      "description": "Classifies incoming support tickets.",
      "protocols": ["mcp"],
      "capabilities": [
        {"name": "classify_ticket", "type": "tool"},
        {"name": "suggest_priority", "type": "tool"}
      ],
      "href": "/ad/r/201"
    },
    {
      "agent": "knowledge-lookup",
      "base": "https://agents.example.com/kb",
      "description": "Searches internal knowledge base.",
      "protocols": ["mcp"],
      "capabilities": [
        {"name": "search_kb", "type": "tool"}
      ],
      "href": "/ad/r/202"
    }
  ]
}
```

B.3. Filtered Lookup with Pagination

A client enumerates MCP agents that expose tool capabilities, one per page.

```
GET /ad/l?protocol=mcp&cap_type=tool&count=1&page=0 HTTP/1.1
Host: ad.example.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json
Link: </ad/l?protocol=mcp&cap_type=tool&count=1&page=1>; rel="next"

{
  "agents": [
    {
      "agent": "ticket-classifier",
      "base": "https://agents.example.com/ticket-classifier",
      "description": "Classifies incoming support tickets.",
      "protocols": ["mcp"],
      "capabilities": [
        {"name": "classify_ticket", "type": "tool"},
        {"name": "suggest_priority", "type": "tool"}
      ],
      "href": "/ad/r/201"
    }
  ]
}
```

The client follows the rel="next" link:

```
GET /ad/l?protocol=mcp&cap_type=tool&count=1&page=1 HTTP/1.1
Host: ad.example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "agents": [
    {
      "agent": "knowledge-lookup",
      "base": "https://agents.example.com/kb",
      "description": "Searches internal knowledge base.",
      "protocols": ["mcp"],
      "capabilities": [
        {"name": "search_kb", "type": "tool"}
      ],
      "href": "/ad/r/202"
    }
  ]
}
```

No Link: rel="next" header is present, indicating the end of results.

B.4. Registration Conflict

A second entity attempts to register an agent under an already-claimed name.

```
POST /ad/r?agent=ticket-classifier HTTP/1.1
Host: ad.example.com
Authorization: Bearer <token-of-other-entity>
Content-Type: application/json

{
  "base": "https://attacker.example.org/ticket-classifier",
  "protocols": ["mcp"],
  "capabilities": [{"name": "classify_ticket", "type": "tool"}]
}

HTTP/1.1 409 Conflict
Content-Type: application/problem+json

{
  "type": "https://ad.example.com/errors/agent-name-taken",
  "title": "Agent name already registered",
  "detail": "Agent name 'ticket-classifier' already registered.",
  "status": 409
}
```

Appendix C. Tooling and Integration

The AD uses application/json for all request and response bodies (and application/problem+json for errors). No custom media types or content negotiation is required.

The HTTP/JSON interface is usable from command-line tools (curl and a JSON parser), shell scripts, and CI/CD pipelines without specialized client libraries. The lookup interface can also be exposed as an MCP tool, so that an MCP-compatible client performs agent discovery through its existing tool-calling mechanism.

Appendix D. Implementation Status

(Boilerplate as per Section 2.1 of [RFC7942]:)

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation

here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

D.1. ad.jaime.win

Organization: Ericsson

Implementation: https://ad.jaime.win (https://ad.jaime.win)

Description: A public AD instance deployed as a Cloudflare Worker with a Durable Object for persistent storage. Implements the registration, lookup, and soft-state lifecycle interfaces defined in this document. Source code is available at the GitHub repository linked from the deployment.

Level of Maturity: Prototype

Coverage: All features specified in this document: well-known URI discovery (URI Template lookup), agent registration (create, update, replace, delete), agent lookup with filtering and Link-header pagination, soft-state expiry, and RFC 9457 error responses.

Version Compatibility: draft-jimenez-agent-directory-00

Licensing: MIT

Contact: Jaime Jimenez (jaime@iki.fi)

Appendix E. Acknowledgments

The AD design is directly based on the CoRE Resource Directory [RFC9176] by Christian Ams端ss, Zach Shelby, Michael Koster, Carsten Bormann, and Peter van der Stok.

Appendix F. Mapping from CoRE RD to AD

The following table summarizes the conceptual mapping between CoRE RD concepts and their AD equivalents.

CoRE RD (RFC 9176)	AD	Notes
Endpoint (EP)	Agent	Software entity that registers
Resource link	Capability	Structured JSON instead of link-format
/.well-known/core	/.well-known/ad	Discovery entry point
Registration (POST /rd)	Registration (POST /ad/r)	Same soft-state model
Registration resource	Registration resource	Same lifecycle
Lifetime (lt)	Lifetime (lt)	Same semantics, different default
Endpoint name (ep)	Agent name (agent)	Same uniqueness constraint
Resource lookup	Capabilities embedded in agent registration	Capabilities are nested in the registration, not independent resources
Endpoint lookup	GET on the lookup endpoint	Structured filters instead of link-format query
Commissioning Tool (CT)	Commissioning Tool (CT)	Third-party registration
application/link-format	application/json	Representation format

Table 2

Author’s Address

Jaime Jimenez
Ericsson
Email: jaime@iki.fi