Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 24 September 2026

Y. Choi
AIEndpoint

23 March 2026

The AI Discovery Endpoint: A Structured Mechanism for AI Agent Service Discovery and Capability Exposure
draft-aiendpoint-ai-discovery-00

Abstract

This document defines a lightweight mechanism by which web services expose a machine-readable description of their capabilities to autonomous AI agents. The mechanism consists of a well-known resource, served at "/.well-known/ai", that returns a structured JSON document describing the service's identity, available actions, authentication requirements, and operational hints optimized for large language model (LLM) token efficiency.

The specification addresses the absence of a standardized method for AI agents to programmatically discover what a web service can do and how to invoke its capabilities, without resorting to parsing human-oriented documentation or HTML content.

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 24 September 2026.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Requirements Language 4
1.2. Terminology 4
2. The AI Discovery Resource 5
2.1. Resource Location 5
2.2. Access Method 5
2.3. Media Type 6
2.4. Character Encoding 6
3. Response Schema 6
3.1. Top-Level Structure 6
3.2. The "service" Object 7
3.3. The "capabilities" Array 8
3.4. The "auth" Object 10
3.5. The "token_hints" Object 10
3.6. The "rate_limits" Object 11
3.7. The "meta" Object 11
4. Processing Rules 12
4.1. Fetching 12
4.2. Caching 12
4.3. Error Handling 13
4.4. Version Negotiation 13
4.5. Size Limits 14
5. Token Efficiency Considerations 14
6. Security Considerations 15
6.1. Information Disclosure 15
6.2. Authentication Credential Exposure 15
6.3. Abuse by Automated Agents 15
6.4. Content Integrity 16
6.5. Denial of Service 16
7. IANA Considerations 16
7.1. Well-Known URI Registration 16
8. Examples 17
8.1. Minimal Conformant Response 17
8.2. Full-Featured Response 17
8.3. Service with No Authentication 18
9. References 20
9.1. Normative References 20

9.2. Informative References 21
Appendix A. Comparison with Existing Approaches 21
A.1. robots.txt (RFC 9309) 22
A.2. OpenAPI / Swagger 22
A.3. Model Context Protocol (MCP) 22
A.4. llms.txt 22
A.5. Summary 23
Appendix B. JSON Schema 23
Appendix C. Design Rationale 23
C.1. Why a Well-Known URI? 23
C.2. Why JSON over Other Formats? 24
C.3. Why Compact Parameter Descriptions? 24
C.4. Why No Request/Response Examples? 24
Acknowledgements 25
Author's Address 25

1. Introduction

The rapid proliferation of autonomous AI agents -- software systems powered by large language models (LLMs) that perform tasks on behalf of users -- has created a fundamental interoperability gap on the World Wide Web. While the web has well-established mechanisms for communicating with human users (HTML) and search engine crawlers (robots.txt [RFC9309], sitemaps), no equivalent standard exists for AI agents seeking to discover and invoke the capabilities of web services.

Today, when an AI agent attempts to interact with a web service, it must resort to one or more of the following strategies:

* Parsing HTML pages designed for human consumption, extracting structured meaning from unstructured content at significant computational cost (typically 8,000-15,000 LLM tokens per page).

* Reading API documentation scattered across multiple pages, often requiring the agent to navigate hyperlinks, interpret natural language descriptions, and resolve ambiguities.

* Using pre-configured tool definitions maintained by the agent's developer, which become stale as services evolve and cannot scale to the breadth of services on the web.

* Attempting trial-and-error API calls, which waste computational resources and risk triggering rate limits or unintended side effects.

These approaches are inefficient, error-prone, and fundamentally unscalable. A single service discovery interaction that could be accomplished in 200-500 LLM tokens with a structured response instead requires 3,000-15,000 tokens of unstructured content parsing -- a 10x to 30x overhead.

This document defines the AI Discovery Endpoint, a mechanism whereby a web service publishes a single, structured JSON document at a well-known location. This document describes the service's identity, enumerates its capabilities as discrete invocable actions, specifies authentication requirements, and provides operational hints that enable AI agents to interact with the service efficiently.

The design prioritizes:

* Simplicity: A developer SHOULD be able to implement a conformant endpoint in under 30 minutes using any web framework.

* Token efficiency: The response format is optimized to convey maximum information in minimum LLM tokens.

* Universality: The mechanism requires only HTTP and JSON -- no custom protocols, transport layers, or runtime dependencies.

* Backward compatibility: Future versions of this specification will extend the schema additively; fields defined in this version will not be removed or have their semantics altered.

1.1. 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.

1.2. Terminology

AI Agent: An autonomous software system, typically powered by a large language model, that performs tasks on behalf of a user by discovering and invoking web services programmatically.

AI Discovery Document: The JSON document served at the AI Discovery Endpoint, conforming to the schema defined in this specification.

Capability: A discrete, invocable action that a web service exposes to AI agents, described by an identifier, an HTTP endpoint, a method, and parameter/return specifications.

Service Provider: The operator of a web service that publishes an AI Discovery Document.

Token: The fundamental unit of text processing in a large language model. The efficiency of AI agent interactions is measured in tokens consumed.

2. The AI Discovery Resource

2.1. Resource Location

A service provider that implements this specification MUST serve the AI Discovery Document at the following well-known URI [RFC8615]:

/.well-known/ai

The complete URI is constructed by appending "/.well-known/ai" to the authority (scheme and host) of the service. For example, for a service at "https://example.com", the AI Discovery Document is located at:

https://example.com/.well-known/ai

A service provider MAY additionally serve an identical copy of the AI Discovery Document at the path "/ai" on the same authority, as a convenience alias. When both locations are served, the content MUST be identical. An AI agent that receives differing content from the two locations MUST treat the "/.well-known/ai" response as authoritative.

The AI Discovery Document MUST NOT be served at a path that includes query parameters or fragment identifiers. The resource is a single, complete document.

2.2. Access Method

The AI Discovery Document MUST be accessible via an HTTP GET request [RFC9110].

The server MUST NOT require authentication to access the AI Discovery Document itself. The purpose of the document is to enable discovery; requiring authentication to discover what authentication is needed creates a circular dependency.

The server SHOULD respond to requests within 3 seconds under normal operating conditions. AI agents MAY treat responses exceeding 10 seconds as unavailable.

The server MUST respond with HTTP status code 200 (OK) when the AI Discovery Document is available and the request is valid.

If the server does not implement this specification, it SHOULD respond with HTTP status code 404 (Not Found). An AI agent receiving a 404 response MUST conclude that the service does not publish an AI Discovery Document.

The server MAY respond with HTTP status code 301 (Moved Permanently) or 302 (Found) to redirect to another location. An AI agent MUST follow redirects up to a maximum of 5 consecutive redirections. The agent MUST NOT follow redirects that change the scheme from "https" to "http".

2.3. Media Type

The AI Discovery Document MUST be served with the media type "application/json" [RFC8259].

Servers SHOULD include the "charset=utf-8" parameter:

```
Content-Type: application/json; charset=utf-8
```

2.4. Character Encoding

The AI Discovery Document MUST be encoded in UTF-8 [RFC3629]. This ensures consistent handling of non-ASCII characters in service descriptions, capability names, and other textual fields, which is particularly important for services operating in multilingual environments.

3. Response Schema

The AI Discovery Document is a single JSON object [RFC8259] containing the fields defined in this section.

3.1. Top-Level Structure

The top-level JSON object MUST contain the following fields:

aiendpoint: (string, REQUIRED) The version of this specification to which the document conforms. For documents conforming to this specification, the value MUST be "1.0".

service: (object, REQUIRED) An object describing the identity of the service. See Section 3.2.

capabilities: (array, REQUIRED) A non-empty array of capability

objects, each describing a discrete action the service exposes to AI agents. See Section 3.3.

The top-level JSON object MAY contain the following fields:

auth: (object, OPTIONAL) An object describing the authentication requirements of the service. See Section 3.4.

token_hints: (object, OPTIONAL) An object providing hints for AI agents on how to reduce token consumption when invoking the service's capabilities. See Section 3.5.

rate_limits: (object, OPTIONAL) An object describing the rate limiting policy of the service. See Section 3.6.

meta: (object, OPTIONAL) An object containing metadata about the AI Discovery Document itself. See Section 3.7.

A conformant AI Discovery Document MUST NOT contain fields not defined in this specification at the top level. This constraint ensures that future versions of the specification can introduce new top-level fields without conflicting with proprietary extensions. Service providers requiring custom metadata SHOULD use the "meta" object for this purpose.

3.2. The "service" Object

The "service" object describes the identity and classification of the service. It MUST contain the following fields:

name: (string, REQUIRED) The human-readable name of the service. This value SHOULD be the commonly recognized name of the service (e.g., "GitHub", "Stripe", "OpenWeatherMap"). The value MUST be between 1 and 100 characters in length.

description: (string, REQUIRED) A concise, factual description of what the service does. This description is intended for machine consumption and SHOULD be written in a declarative style that prioritizes clarity over marketing language. The value MUST be between 1 and 300 characters in length.

Authors SHOULD aim for descriptions under 200 characters to optimize token consumption.

The "service" object MAY contain the following fields:

category: (array of strings, OPTIONAL) One or more category labels

that classify the service for discovery purposes. The following category values are defined in this version of the specification: "productivity", "ecommerce", "finance", "news", "weather", "maps", "search", "data", "communication", "calendar", "storage", "media", "health", "education", "travel", "food", "government", "developer".

Service providers SHOULD select from the above list to maximize interoperability with discovery systems. Future versions of this specification MAY define additional categories. A discovery system encountering an unrecognized category value SHOULD ignore it without error.

The array MUST contain at least one element if present, and MUST NOT contain duplicate values.

language: (array of strings, OPTIONAL) The natural languages supported by the service, expressed as BCP 47 [RFC5646] language tags. If omitted, the default value is ["en"].

The array MUST contain at least one element if present, and MUST NOT contain duplicate values.

3.3. The "capabilities" Array

The "capabilities" array is the core of the AI Discovery Document. Each element describes a single, discrete action that an AI agent can invoke.

The array MUST contain at least one element.

Service providers SHOULD limit the array to capabilities that are meaningful for AI agent consumption. Internal endpoints, administrative functions, and endpoints that require complex multi-step human interaction SHOULD be omitted.

Each capability object MUST contain the following fields:

id: (string, REQUIRED) A unique identifier for this capability within the service. The identifier MUST match the regular expression ^[a-z][a-z0-9_]*$ (lowercase ASCII letters, digits, and underscores, beginning with a letter). The identifier MUST be between 1 and 64 characters in length.

The identifier serves as a stable reference for the capability. Service providers SHOULD NOT change the identifier of an existing capability, as AI agents and external systems may reference it.

description: (string, REQUIRED) A concise description of what this capability does. The value MUST be between 1 and 200 characters in length.

This description is the primary mechanism by which an AI agent determines whether a capability is relevant to its task. Authors SHOULD write descriptions that are specific and actionable.

endpoint: (string, REQUIRED) The URI or URI path at which this capability is accessible. The value MUST be non-empty.

If the value begins with "/" (U+002F), it is a relative path and MUST be resolved against the authority of the AI Discovery Document's URI. If the value is an absolute URI (beginning with a scheme), it is used as-is. This allows services to expose capabilities hosted on different domains or subdomains.

method: (string, REQUIRED) The HTTP method used to invoke this capability. The value MUST be one of: "GET", "POST", "PUT", "DELETE", "PATCH".

Each capability object MAY contain the following fields:

params: (object, OPTIONAL) A JSON object describing the parameters accepted by this capability's endpoint. Each key is a parameter name, and each value is a string describing the parameter in a compact, structured format.

The parameter description string SHOULD follow this pattern:

<type>, <requirement>[, <constraints>] [-- <description>]

Where:

* <type> is the data type: "string", "integer", "number", "boolean", or "array"

* <requirement> is "required" or "optional"

* <constraints> are optional qualifiers such as "default 10", "max 50", "min 1", or enumerated values separated by "|"

* <description> is an optional natural language description, separated by an em dash or double hyphen (--)

returns: (string, OPTIONAL) A compact description of the response structure. The value MUST NOT exceed 300 characters in length.

Authors SHOULD use a concise structural notation that conveys the shape of the response, e.g., "products[] {id, name, price, stock}".

3.4. The "auth" Object

The "auth" object describes how an AI agent must authenticate when invoking the service's capabilities.

When present, the "auth" object MUST contain the following field:

type: (string, REQUIRED) The authentication mechanism required by the service. The value MUST be one of:

- "none": No authentication is required. All capabilities are publicly accessible.
- "apikey": Authentication via a static API key, typically passed in an HTTP header.
- "bearer": Authentication via a bearer token (e.g., JWT), passed in the Authorization header as defined in [RFC6750].
- "oauth2": Authentication via the OAuth 2.0 framework [RFC6749].

The "auth" object MAY contain the following fields:

header: (string, OPTIONAL) The name of the HTTP header in which the authentication credential is passed. This field is most useful for "apikey" authentication where the header name varies across services (e.g., "X-API-Key", "Api-Key", "Authorization"). For "bearer" authentication, the header is implicitly "Authorization" and this field MAY be omitted.

docs: (string, OPTIONAL) A URI pointing to human-readable documentation for the service's authentication process.

If the "auth" object is omitted entirely, an AI agent SHOULD assume that authentication may be required. Service providers are therefore RECOMMENDED to include the "auth" object even when no authentication is required, with type set to "none".

3.5. The "token_hints" Object

The "token_hints" object provides signals to AI agents about features the service supports to reduce the token cost of capability invocations.

All fields in the "token_hints" object are boolean and OPTIONAL. If the object is omitted, all fields default to false.

compact_mode: (boolean, default false) If true, the service's capability endpoints accept a query parameter "compact=true" that causes the response to include only essential fields, omitting verbose or supplementary data.

field_filtering: (boolean, default false) If true, the service's capability endpoints accept a query parameter "fields" whose value is a comma-separated list of field names. The response includes only the specified fields.

delta_support: (boolean, default false) If true, the service's capability endpoints accept a query parameter "since" whose value is an ISO 8601 timestamp. The response includes only records created or modified after the specified timestamp.

3.6. The "rate_limits" Object

The "rate_limits" object informs AI agents of the service's rate limiting policy, enabling agents to plan their request patterns to avoid throttling.

requests_per_minute: (integer, OPTIONAL) The maximum number of requests an AI agent may make per minute under the default access tier. The value MUST be a positive integer. This is an advisory limit. AI agents SHOULD use this value to pace their requests.

agent_tier_available: (boolean, OPTIONAL) If true, the service offers a higher rate limit tier specifically for AI agents. Details of how to access this tier are available at the URI specified in "auth.docs".

3.7. The "meta" Object

The "meta" object contains metadata about the AI Discovery Document itself (not the service it describes). All fields are OPTIONAL.

last_updated: (string, OPTIONAL) The date on which the AI Discovery Document was last modified, expressed in ISO 8601 format (YYYY-MM-DD or YYYY-MM-DDThh:mm:ssZ).

changelog: (string, OPTIONAL) A URI pointing to a changelog or release notes page for the service.

status: (string, OPTIONAL) A URI pointing to a service status page.

AI agents MAY consult this URI before invoking capabilities on a service that appears to be experiencing errors.

4. Processing Rules

This section defines how AI agents and other consumers SHOULD process the AI Discovery Document.

4.1. Fetching

An AI agent attempting to discover the capabilities of a service at a given authority SHOULD perform the following steps:

1. Construct the well-known URI by appending "/.well-known/ai" to the authority (e.g., "https://example.com/.well-known/ai").

2. Issue an HTTP GET request to the constructed URI.

3. If the response status is 200 (OK) and the Content-Type header indicates "application/json", parse the response body as a JSON object.

4. Validate that the parsed object contains the required fields "aiendpoint", "service", and "capabilities" as defined in Section 3.

5. If validation succeeds, the service is considered to have a valid AI Discovery Document. The agent MAY proceed to invoke the service’s capabilities as described.

If the response status is 404 (Not Found), the agent MAY optionally attempt to fetch the resource at "/ai" as a fallback. However, the "/.well-known/ai" location is authoritative, and agents MUST NOT treat a response from "/ai" as valid if the "/.well-known/ai" location returns a different response.

An agent MUST NOT issue requests to any capability endpoint described in the AI Discovery Document without first confirming that the document is valid and that the capability’s method, endpoint, and parameters are well-formed.

4.2. Caching

The AI Discovery Document is expected to change infrequently. Service providers SHOULD include appropriate HTTP caching headers [RFC9111] in the response. A Cache-Control max-age value of 86400 (24 hours) is RECOMMENDED for most services.

```
Cache-Control: public, max-age=86400
```

AI agents SHOULD respect standard HTTP caching semantics. When the "meta.last_updated" field is present, an agent MAY compare it against a previously stored value to determine whether the document has changed.

4.3. Error Handling

AI agents MUST handle the following error conditions gracefully:

HTTP 404 (Not Found): The service does not implement this specification. The agent MUST NOT retry the request for the same authority within a reasonable period (RECOMMENDED: 24 hours).

HTTP 5xx (Server Error): The service may be temporarily unavailable. The agent MAY retry after an appropriate delay, using exponential backoff with a maximum of 3 retry attempts.

HTTP 429 (Too Many Requests): The agent has exceeded the service's rate limit. The agent MUST respect the Retry-After header if present.

Invalid JSON: If the response body cannot be parsed as valid JSON, the agent MUST treat the service as not implementing this specification.

Schema Violation: If the parsed JSON does not contain the required fields, the agent SHOULD treat the document as invalid. An agent MAY attempt to use partially valid documents at its own risk.

4.4. Version Negotiation

This specification defines version "1.0" of the AI Discovery Document. The "aiendpoint" field identifies the version.

An AI agent that encounters an "aiendpoint" value it does not recognize SHOULD still attempt to parse the document using the rules of the highest version it supports. Because this specification mandates additive-only changes across versions, a version 1.0 agent can safely process a version 1.1 document by ignoring unknown fields.

An AI agent MUST NOT reject a document solely because the "aiendpoint" value is higher than the version it supports.

4.5. Size Limits

An AI Discovery Document SHOULD NOT exceed 64 kilobytes in size. Service providers whose capabilities would exceed this limit SHOULD consider splitting their service into logical sub-services, each with its own AI Discovery Document.

An AI agent MAY refuse to process a document that exceeds 256 kilobytes.

5. Token Efficiency Considerations

A primary design goal of this specification is to minimize the number of LLM tokens required for an AI agent to discover and understand a service’s capabilities.

As of this writing, leading LLMs consume approximately 1 token per 4 characters of English text and 1 token per 1-2 characters of JSON structural elements. Non-Latin scripts (e.g., CJK characters) typically consume 1-3 tokens per character.

A well-formed AI Discovery Document for a service with 5 capabilities is expected to consume between 200 and 800 tokens. By contrast, extracting equivalent information from a typical HTML documentation page requires 3,000 to 15,000 tokens -- an overhead factor of 10x to 30x.

This efficiency gain is achieved through several design choices:

* Compact parameter descriptions: Parameters are described in a single string that encodes type, requirement, constraints, and description in a compact notation, rather than a verbose schema format as in OpenAPI.

* Structural response descriptions: The "returns" field uses a concise notation (e.g., "products[] {id, name, price}") rather than a full JSON Schema definition.

* Absence of examples: Unlike OpenAPI, this specification does not include example request/response pairs in the discovery document. AI agents are expected to infer usage from parameter and return descriptions.

* Fixed vocabulary: Category values and authentication types use a closed vocabulary, eliminating the need for agents to interpret free-form text in these fields.

6. Security Considerations

The AI Discovery Document inherently exposes information about a service's internal structure and capabilities. Service providers MUST carefully consider the security implications of this disclosure.

6.1. Information Disclosure

The AI Discovery Document reveals the existence, paths, methods, and parameter structures of API endpoints. While this information may already be available through documentation or reverse engineering, the structured, machine-readable format significantly lowers the barrier for automated reconnaissance.

Service providers MUST NOT include endpoints in the "capabilities" array that they do not intend to be publicly discoverable. Administrative endpoints, internal APIs, and endpoints under development MUST be excluded.

6.2. Authentication Credential Exposure

The "auth" object describes the authentication mechanism but MUST NOT contain actual credentials, secrets, tokens, or keys. The "header" field specifies the name of the header in which a credential is passed, not the credential itself.

An AI agent that stores or caches AI Discovery Documents MUST ensure that no credentials are written into the cached document, even if the agent possesses credentials for the service.

When the "auth.type" is "none", the service provider asserts that no authentication is required. Service providers MUST NOT use "none" for endpoints that handle sensitive data, perform write operations, or incur costs.

6.3. Abuse by Automated Agents

The AI Discovery Document facilitates automated interaction with web services. This creates a potential vector for abuse, including:

* Automated enumeration of capabilities across many services for reconnaissance purposes.

* Rapid, programmatic invocation of capabilities at scale, potentially exceeding the service's intended usage patterns.

* Exploitation of capabilities that were not designed with automated consumption in mind.

Service providers MUST implement appropriate rate limiting on all capability endpoints exposed through the AI Discovery Document. Service providers MUST NOT rely solely on agents' voluntary compliance with advertised rate limits.

Service providers SHOULD monitor traffic patterns to capability endpoints and implement anomaly detection for automated abuse.

6.4. Content Integrity

An AI agent that fetches an AI Discovery Document over an insecure channel risks receiving a modified or malicious document. A tampered document could redirect the agent to attacker-controlled endpoints or alter parameter descriptions to induce the agent to send sensitive data to unauthorized parties.

AI agents MUST fetch the AI Discovery Document over HTTPS (TLS) [RFC8446]. Agents MUST NOT fetch the document over plain HTTP. Service providers MUST serve the document over HTTPS.

AI agents SHOULD validate TLS certificates according to [RFC6125] and MUST NOT proceed if certificate validation fails.

6.5. Denial of Service

A malicious or misconfigured AI Discovery Document could describe a very large number of capabilities, extremely long description strings, or deeply nested parameter structures, causing excessive resource consumption in the parsing agent.

AI agents SHOULD enforce the size limits described in Section 4.5 and SHOULD impose their own limits on the number of capabilities processed (RECOMMENDED: no more than 100 capabilities per document) and the length of individual string fields.

7. IANA Considerations

7.1. Well-Known URI Registration

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

URI Suffix: ai

Change Controller: IETF

Specification Document(s): This document

Status: provisional

Related Information: None

8. Examples

8.1. Minimal Conformant Response

```
{
  "aiendpoint": "1.0",
  "service": {
    "name": "SimpleNotes",
    "description": "Create and retrieve plain text notes."
  },
  "capabilities": [
    {
      "id": "create_note",
      "description": "Create a new text note",
      "endpoint": "/api/notes",
      "method": "POST"
    },
    {
      "id": "list_notes",
      "description": "List all notes, newest first",
      "endpoint": "/api/notes",
      "method": "GET"
    }
  ]
}
```

8.2. Full-Featured Response

```
{
  "aiendpoint": "1.0",
  "service": {
    "name": "ExampleShop",
    "description": "Search and browse products.
      Supports keyword search, category filtering,
      and price sorting.",
    "category": ["ecommerce", "search"],
    "language": ["en", "ko"]
  },
  "capabilities": [
    {
      "id": "search_products",
      "description": "Search products by keyword",
      "endpoint": "/api/ai/products/search",
      "method": "GET",
```

```
        "params": {
          "q": "string, required -- search keyword",
          "category": "string, optional -- filter by category",
          "max_price": "number, optional -- max price in USD",
          "sort": "string, optional --
            price_asc|price_desc|relevance,
            default relevance",
          "limit": "integer, optional, default 10, max 50"
        },
        "returns": "products[] {id, name,
          price_usd, stock, category, url}"
      },
      {
        "id": "get_product",
        "description": "Get full details of a product by ID",
        "endpoint": "/api/ai/products/:id",
        "method": "GET",
        "params": {
          "id": "string, required -- product ID"
        },
        "returns": "product {id, name, description,
          price_usd, stock, images[], category, url}"
      }
    ],
    "auth": {
      "type": "apikey",
      "header": "X-API-Key",
      "docs": "https://exampleshop.com/docs/auth"
    },
    "token_hints": {
      "compact_mode": true,
      "field_filtering": true,
      "delta_support": false
    },
    "rate_limits": {
      "requests_per_minute": 60,
      "agent_tier_available": true
    },
    "meta": {
      "last_updated": "2026-03-10",
      "changelog": "https://exampleshop.com/changelog",
      "status": "https://status.exampleshop.com"
    }
  }
```

8.3. Service with No Authentication

```
{
  "aiendpoint": "1.0",
  "service": {
    "name": "WorldWeather",
    "description": "Current weather and 5-day
      forecasts for cities worldwide.",
    "category": ["weather", "data"],
    "language": ["en"]
  },
  "capabilities": [
    {
      "id": "current_weather",
      "description": "Get current weather for a city",
      "endpoint": "/api/weather/current",
      "method": "GET",
      "params": {
        "city": "string, required -- city name",
        "units": "string, optional --
          metric|imperial, default metric"
      },
      "returns": "city {name, country}, condition,
        temp, humidity_pct, wind_speed, updated_at"
    },
    {
      "id": "forecast",
      "description": "Get 5-day weather forecast for a city",
      "endpoint": "/api/weather/forecast",
      "method": "GET",
      "params": {
        "city": "string, required -- city name",
        "days": "integer, optional, default 5, max 5",
        "units": "string, optional --
          metric|imperial, default metric"
      },
      "returns": "city {name, country},
        forecast[] {date, condition, temp_high,
        temp_low, precipitation_pct}"
    }
  ],
  "auth": {
    "type": "none"
  },
  "token_hints": {
    "compact_mode": false,
    "field_filtering": true,
    "delta_support": true
  },
  "rate_limits": {
```

```
      "requests_per_minute": 100,
      "agent_tier_available": false
    },
    "meta": {
      "last_updated": "2026-03-15"
    }
  }
```

9. References

9.1. 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/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[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/info/rfc5646>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[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/info/rfc6750>.

[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/info/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/info/rfc8259>.

[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/info/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/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/info/rfc9110>.

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

9.2. Informative References

[RFC9309] Koster, M., Illyes, G., Zeller, H., and L. Sassman, "Robots Exclusion Protocol", RFC 9309, DOI 10.17487/RFC9309, September 2022, <https://www.rfc-editor.org/info/rfc9309>.

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

[MCP] Anthropic, "Model Context Protocol Specification", 2024, <https://modelcontextprotocol.io>.

[LLMSTXT] Willison, S., "llms.txt -- A Proposal", 2024, <https://llmstxt.org>.

Appendix A. Comparison with Existing Approaches

A.1. robots.txt (RFC 9309)

robots.txt communicates access restrictions to web crawlers. It answers the question "where should you NOT go?" but provides no information about what a service can do or how to interact with it programmatically. The AI Discovery Endpoint is complementary: while robots.txt defines boundaries, the AI Discovery Endpoint defines capabilities within those boundaries.

A.2. OpenAPI / Swagger

OpenAPI provides exhaustive, formal descriptions of REST APIs. It is designed as a complete contract between API provider and consumer, typically running to thousands of lines for a non-trivial service. A typical OpenAPI specification for a service with 10 endpoints consumes 5,000-20,000 tokens when processed by an LLM. The equivalent AI Discovery Document consumes 300-800 tokens.

The AI Discovery Endpoint does not replace OpenAPI. A service MAY publish both, using the AI Discovery Document for initial capability discovery and linking to the OpenAPI specification for agents that require detailed contract information.

A.3. Model Context Protocol (MCP)

MCP defines a runtime protocol for connecting AI agents to external tools and data sources. It requires a dedicated server process, a custom transport layer, and framework-specific integration. The AI Discovery Endpoint operates at a different layer: it is a static discovery mechanism with no runtime dependencies. A service’s AI Discovery Document can be used to automatically generate an MCP server configuration, making the two approaches complementary.

A.4. llms.txt

llms.txt [LLMSTXT] proposes a convention for placing a plain text file at the root of a website containing a natural language summary for LLMs. While simple to implement, it is unstructured, not machine-queryable, and does not describe invocable actions. An LLM can read an llms.txt file to understand what a service is, but cannot determine from it how to programmatically interact with the service.

A.5. Summary

Criterion	/ai	OpenAPI	MCP	llms.txt
Discovery focus	Yes	No	No	Partial
Token efficient	Yes	No	N/A	Partial
Machine-queryable	Yes	Yes	Yes	No
Actions described	Yes	Yes	Yes	No
No infrastructure	Yes	Yes	No	Yes
Implementation time	<30 min	Hours	Hours	<10 min

Table 1

Appendix B. JSON Schema

A formal JSON Schema (draft 2020-12) for the AI Discovery Document is maintained at:

https://www.aiendpoint.dev/spec/v1/schema.json

This schema provides machine-verifiable validation of AI Discovery Documents and is the normative reference for field types, constraints, and enumerated values defined in this specification. The schema is versioned in parallel with this specification; version "1.0" of the schema corresponds to this document.

Appendix C. Design Rationale

C.1. Why a Well-Known URI?

The Well-Known URI mechanism [RFC8615] provides a standardized namespace for resources that need to be located without prior knowledge of a site’s URL structure. This is precisely the use case for AI agent discovery: an agent needs to find the AI Discovery Document given only the service’s hostname. The alternative -- requiring services to register their AI Discovery Document location in a central registry -- would create a dependency on a third-party service and limit adoption.

C.2. Why JSON over Other Formats?

JSON is the de facto standard for structured data exchange on the web. Every mainstream programming language includes JSON parsing in its standard library. Every LLM is trained on extensive JSON data and can parse it reliably.

Alternatives considered:

* YAML: More human-readable but introduces parsing complexity and ambiguity.

* XML: More verbose, imposing higher token costs.

* Protocol Buffers / MessagePack: Efficient but not human-readable.

* Plain text (like llms.txt): Simple but unstructured, preventing reliable machine parsing.

C.3. Why Compact Parameter Descriptions?

The decision to represent parameter descriptions as single-line strings (e.g., "string, required -- search keyword") rather than structured objects (as in OpenAPI's Parameter Object) was driven by token efficiency. A structured representation requires approximately 8-12 JSON tokens for structural elements alone. The compact string format eliminates this overhead while remaining parseable by both humans and LLMs.

C.4. Why No Request/Response Examples?

This specification deliberately omits examples from the discovery document for two reasons:

1. Token cost: Examples can easily double the size of a capability description without adding proportional informational value for an LLM, which can infer usage from parameter types and descriptions.

2. Maintenance burden: Examples that fall out of sync with the actual API behavior are worse than no examples, as they cause agents to generate incorrect requests.

Acknowledgements

The design of this specification was informed by the long history of web standards that enable automated interaction with web resources, including the Robots Exclusion Protocol [RFC9309], the Sitemaps protocol, the OpenAPI Specification [OPENAPI], and the Model Context Protocol [MCP].

The authors thank the early adopters and contributors to the AIEndpoint open-source project for their feedback on the specification design.

Author’s Address

Yeongjae Choi
AIEndpoint
Email: bejoyfuuul@gmail.com
URI: https://www.aiendpoint.dev