Individual Submission
Internet-Draft
Intended status: Standards Track
Expires: 19 September 2026

T. Nederveld
Ironstead Group, LLC.

18 March 2026

Agent Definition Language (ADL)
draft-nederveld-adl-02

Abstract

The Agent Definition Language (ADL) provides a standard JSON-based format for describing AI agents. An ADL document declares an agent's identity, capabilities, tools, permissions, security requirements, data classification, and runtime configuration in a single, machine-readable artifact. ADL enables discovery, interoperability, deployment, and lifecycle management of AI agents across diverse platforms and runtimes. This document defines the structure of ADL documents, the semantics of their members, conformance requirements for implementations, and the registration of the application/adl+json media type.

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 19 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights

and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 4
1.1. Purpose 4
1.2. Goals 5
1.3. Design Model 5
1.4. Relationship to Other Specifications 6
2. Requirements Language 6
3. Terminology 7
4. Document Structure 8
4.1. Media Type 8
4.2. Top-Level Object 8
4.3. Extension Mechanism 9
4.4. Pattern Matching 10
5. Core Members 10
5.1. ADL Specification 11
5.2. $schema 11
5.3. Name 11
5.4. Description 11
5.5. Version 11
5.6. Lifecycle 12
5.6.1. status 12
5.6.2. effective_date 13
5.6.3. sunset_date 13
5.6.4. successor 13
6. Agent Identity 13
6.1. Id 13
6.2. Provider 14
6.3. Cryptographic Identity 15
6.4. Discovery 15
7. Model Configuration 16
7.1. Model 16
7.2. System Prompt 17
7.2.1. Template Variable Syntax 17
8. Capabilities 18
8.1. Tools 18
8.2. Resources 19
8.3. Prompts 20
9. Permissions 20
9.1. Permissions Model 20
9.1.1. Conflict Resolution 21
9.2. Network 21
9.3. Filesystem 22

9.4. Environment 22
9.5. Execution 22
9.6. Resource Limits 22
10. Security 23
10.1. Authentication 23
10.2. Encryption 23
10.3. Attestation 23
10.4. Data Classification 24
10.4.1. High-Water Mark Rule 24
10.4.2. sensitivity 25
10.4.3. categories 25
10.4.4. retention 26
10.4.5. handling 26
10.4.6. Profile Extensions 27
11. Runtime Behavior 28
11.1. Input Handling 28
11.2. Output Handling 28
11.3. Tool Invocation 28
11.4. Error Handling 29
12. Metadata 30
12.1. Authors 30
12.2. License 30
12.3. Documentation 30
12.4. Repository 30
12.5. Tags 30
12.6. Example 30
13. Profiles 31
13.1. Profile Schema Composition 31
13.2. Multi-Profile Composition 32
13.3. Profile Dependencies 32
13.4. Vendor Profiles 33
13.5. Standard Profile Registration 34
13.6. Example 35
14. Processing ADL Documents 35
14.1. Parsing 35
14.2. Validation 35
14.3. Unknown Members 37
15. Interoperability 37
15.1. A2A Agent Card Generation 37
15.2. MCP Server Configuration 37
15.3. OpenAPI Integration 37
16. Errors 37
16.1. Error Format 37
16.2. Error Codes 38
16.3. Error Source Examples 39
17. IANA Considerations 40
17.1. Media Type 40
17.2. Profile Registry 42

17.3. URN Namespace 45
17.4. Well-Known URI 45
18. Security Considerations 46
18.1. Document Integrity 46
18.2. Sensitive Data in ADL Documents 46
18.3. Template Injection 47
18.4. Information Disclosure 47
18.5. Resource Exhaustion 48
18.6. Pattern Matching Abuse 48
18.7. URI Reference Attacks (SSRF) 49
18.8. Canonicalization Attacks 49
18.9. Privacy Considerations 50
18.10. Privilege Escalation 50
18.11. Cross-Origin and Supply Chain Concerns 51
18.12. Permission Model and Defense in Depth 52
19. References 52
19.1. Normative References 52
19.2. Informative References 53
Appendix A. JSON Schema 54
Appendix B. Examples 54
Appendix C. Profiles 55
C.1. Available Profiles 55
Appendix D. ABNF Grammar 55
D.1. Cross-Reference Summary 56
Acknowledgments 57
Author’s Address 57

1. Introduction

1.1. Purpose

The Agent Definition Language (ADL) provides a standard format for describing AI agents. ADL documents are JSON objects that describe an agent’s identity, capabilities, tools, permissions, and runtime requirements. This specification describes the structure of ADL documents, the semantics of their members, and conformance requirements for implementations.

ADL serves a similar role for AI agents that OpenAPI serves for REST APIs, AsyncAPI for event-driven architectures, and WSDL for web services. It enables:

* *Discovery:* Agents can be found and understood programmatically.

* *Interoperability:* Agents can interact with tools, resources, and other agents using a common description format.

* *Deployment:* Runtime environments can provision and configure agents based on declared requirements.

* *Security:* Permission boundaries and security requirements are explicitly declared and enforceable.

* *Lifecycle:* Agents can be versioned, tracked through operational states, and managed across their entire lifecycle from draft to retirement.

1.2. Goals

* *Portable:* ADL documents describe agents independent of any specific runtime, platform, or provider.

* *Interoperable:* ADL documents can be transformed into other formats (A2A Agent Cards, MCP configurations) and consumed by diverse tooling.

* *Extensible:* ADL supports profiles that add domain-specific requirements without changing the core specification.

* *Secure:* Permission boundaries, authentication, and security constraints are first-class concepts.

* *Machine-readable:* ADL documents are validated against JSON Schema and can be processed programmatically.

* *Human-friendly:* Clear naming conventions and structures that are easy to read and author.

1.3. Design Model

An ADL document functions as a *passport* for an AI agent. It carries the declarations that a counterparty -- peer agent, gateway, orchestrator, registry, or human operator -- needs to make a trust decision: identity, capabilities, permissions, security posture, and governance signals.

The passport model establishes two principles:

1. *Self-contained trust signals.* An ADL document *MUST* contain enough information for a counterparty to evaluate whether to interact with the agent, without requiring access to external systems.

2. *Separation of declaration from operations.* Operational detail that changes independently of the agent's declared behavior -- escalation contacts, audit schedules, evaluation reports, deployment logs -- belongs in external records (e.g., a governance record in a registry), not in the passport. Profiles *MAY* define linking members (e.g., governance_record_ref) that reference such records by stable URI.

This separation ensures that: - The passport remains compact for agent-to-agent and agent-to-gateway interactions. - Operational changes (personnel rotation, policy updates) do not require re-issuing the passport. - Internal operational detail is not exposed to external counterparties.

1.4. Relationship to Other Specifications

ADL builds upon and interoperates with:

* **JSON [RFC8259] -- ADL documents are valid JSON.

* *JSON Schema* [JSON-SCHEMA] -- ADL documents are validated against JSON Schema; tool parameters use JSON Schema for types.

* *A2A Protocol* [A2A] -- ADL documents can generate A2A Agent Cards.

* *Model Context Protocol (MCP)* [MCP] -- ADL documents can generate MCP server configurations; tools, resources, and prompts align with MCP primitives.

* *OpenAPI* [OPENAPI] -- ADL can reference OpenAPI specifications for HTTP-based tools.

* *W3C DIDs* [W3C.DID] / *Verifiable Credentials* [W3C.VC] -- ADL supports DIDs for cryptographic identity and VCs for attestations.

2. 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.

3. Terminology

The terms "AI agent", "AI system", "autonomy", and "automation" are used in this document consistent with their definitions in [ISO-22989]. Where this specification narrows an ISO/IEC 22989 term, the narrower definition below takes precedence.

Term	Definition
ADL document	A JSON object that conforms to this specification.
agent	An AI agent [ISO-22989] further scoped as an AI system [ISO-22989] that operates within boundaries declared by an ADL document. An agent senses and responds to its environment and takes actions to achieve its goals, subject to the permissions and constraints expressed in its ADL document.
AI system	An engineered system that generates outputs such as content, forecasts, recommendations, or decisions for a given set of human-defined objectives [ISO-22989].
model	The AI model (e.g., large language model) that powers an agent’s reasoning. In [ISO-22989] terms, a model is the learned computational artifact within an AI system.
tool	A function or capability that an agent can invoke to perform an action or retrieve information (equivalent to "function" in function-calling and "tool" in [MCP]).
resource	A data source that an agent can read from (e.g., vector store, knowledge base, file system).
prompt	A predefined prompt template that an agent can use.
profile	A set of additional requirements and members that extend the core ADL specification for specific domains.
permission domain	A category of system access (network, filesystem, etc.) that defines operational boundaries for an agent.

runtime	The system or environment that executes an agent based on its ADL definition.
autonomy	The characteristic of a system that is capable of modifying its intended domain of use or goal without external intervention, control, or oversight [ISO-22989]. ADL expresses the degree of permitted autonomy through the autonomy member and governance profile tiers.

Table 1

4. Document Structure

4.1. Media Type

* ADL documents use the media type *application/adl+json*.

* ADL documents *MUST* be encoded in UTF-8.

* ADL documents *MUST* be valid JSON [RFC8259].

* Member names *MUST* use *snake_case* (lowercase with underscores).

* All timestamps *MUST* be ISO 8601 strings with timezone (e.g., "2026-02-15T14:30:00Z").

* All URIs *MUST* conform to [RFC3986].

* *YAML authoring and JSON canonical form:* YAML is an authoring convenience; JSON is the canonical wire format. When an ADL document is authored in YAML, implementations *MUST* convert it to JSON for processing and validation. The media type application/adl+json applies to the JSON canonical form.

4.2. Top-Level Object

An ADL document *MUST* be a single JSON object.

Required members:

* adl_spec (Section 5.1)

* name (Section 5.3)

* description (Section 5.4)

* version (Section 5.5)

* data_classification (Section 10.4)

Optional members:

* $schema, id, provider, cryptographic_identity, lifecycle, model, system_prompt, tools, resources, prompts, permissions, security, runtime, metadata, profiles

An ADL document *MUST NOT* contain members not defined by this specification, a declared profile, or the extension mechanism.

4.3. Extension Mechanism

* *Profiles:* Add domain-specific requirements and members; declared in profiles. See Section 13.

* *Extensions object:* Custom vendor data without a full profile. An extensions member *MAY* appear at any object level within an ADL document. Within extensions, vendor data is grouped under reverse-domain namespace keys.

Vendor namespace keys *MUST* use reverse-domain notation with at least two dot-separated segments (e.g., com.acme, io.anthropic, org.example.research). Keys *MUST* conform to the vendor-key production in Appendix D. Single-segment keys (e.g., acme) and uppercase keys (e.g., COM.ACME) are invalid.

Implementations *MUST* preserve extensions members when round-tripping ADL documents. Implementations *MAY* ignore the contents of extensions. Implementations *MUST NOT* reject documents containing extensions with unknown vendor namespaces.

The member name extensions is reserved at every object level in an ADL document. Implementations *MUST NOT* define non-extension semantics for the extensions member.

Example:

json { "adl_spec": "0.2.0", "name": "Invoice Processor", "version": "2.0.0", "description": "Processes and routes invoices.", "data_classification": { "sensitivity": "confidential", "extensions": { "com.acme": { "data_tier": "gold", "retention_override_approved": true } } }, "model": { "name": "acme-large-2024", "extensions": { "com.acme": { "model_tier": "premium", "cost_per_1k_tokens": 0.03 } } }, "extensions": { "com.acme": { "internal_id": "inv-proc-007", "cost_center": "engineering" } } }

4.4. Pattern Matching

Several ADL members use patterns to specify allowed or denied values. ADL defines a minimal pattern syntax based on a subset of glob matching rules. The following constructs are supported:

1. *Literal match.* A string with no wildcard characters matches only itself. Matching is case-sensitive unless the underlying system is case-insensitive (e.g., Windows filesystem paths).

2. *Single-segment wildcard (*).* The * character matches zero or more characters within a single segment. The segment boundary depends on context:

   * *Host patterns* (Section 9.2): segments are separated by . (dot). * does not match dots. *.example.com matches api.example.com but does not match deep.sub.example.com.

   * *Environment variable patterns* (Section 9.4): * matches any characters in the variable name. APP_* matches APP_PORT and APP_HOST.

   * *Command patterns* (Section 9.5): * matches any characters in the command name.

3. *Multi-segment wildcard (**).* The ** sequence matches zero or more path segments including separators. Valid only in filesystem path patterns (Section 9.3). /data/** matches /data/, /data/foo, and /data/foo/bar/baz. ** *MUST NOT* appear in host patterns, environment variable patterns, or command patterns.

4. *Restrictions.* Patterns *MUST* contain wildcards only in the positions described above. Mid-string wildcards (e.g., foo*bar) are *NOT RECOMMENDED*; implementations *MAY* reject them. A bare * as an entire pattern (matching everything) is valid but *NOT RECOMMENDED* for security-sensitive domains (allowed_hosts, allowed_variables). Implementations *SHOULD* warn when a bare * wildcard is used in permission patterns.

Implementations *MUST* apply patterns using the rules defined in this section. Implementations *MUST NOT* interpret patterns as regular expressions. Formal grammar productions for pattern elements are defined in Appendix D.

5. Core Members

5.1. ADL Specification

Specifies the ADL specification version the document conforms to.

* *REQUIRED.* Value *MUST* be a string in semantic versioning format (MAJOR.MINOR.PATCH). The format *MUST* conform to the semver production in Appendix D.

* Implementations *MUST* reject documents with an unsupported adl_spec version.

* Implementations *SHOULD* support documents with the same MAJOR version and lower or equal MINOR version.

* Pre-release suffixes (e.g., "0.2.0-draft") *MUST NOT* appear in adl_spec values. Only release versions are valid for conformance. Pre-release identifiers *MAY* appear in the agent's own version member (Section 5.5).

Example: "adl_spec": "0.2.0"

5.2. $schema

Optional. URI reference to the JSON Schema for validation. *RECOMMENDED* for JSON documents (enables IDE validation). Canonical schema URI for ADL 0.2: https://adl-spec.org/0.2/schema.json.

5.3. Name

Human-readable name for the agent. *REQUIRED.* Value *MUST* be a non-empty string. For machine identifiers, use id (Section 6.1).

5.4. Description

Human-readable description of the agent's purpose and capabilities. *REQUIRED.* Value *MUST* be a non-empty string. *SHOULD* be sufficient for users to understand what the agent does without examining tool definitions.

5.5. Version

Agent's version. *REQUIRED.* Value *MUST* be a string in semantic versioning format (MAJOR.MINOR.PATCH); the format *MUST* conform to the semver production in Appendix D. Agent version changes *SHOULD* follow SemVer (MAJOR: breaking; MINOR: new capabilities; PATCH: fixes, docs).

5.6. Lifecycle

Operational lifecycle status of the agent. *OPTIONAL.* When present, value *MUST* be an object containing at minimum a status member.

Member	Type	Required	Description
status	string	REQUIRED	Lifecycle state of the agent
effective_date	string	OPTIONAL	ISO 8601 timestamp when current status took effect
sunset_date	string	OPTIONAL	ISO 8601 timestamp for planned or actual retirement
successor	string	OPTIONAL	URI or URN of the replacement agent

Table 2

5.6.1. status

REQUIRED when lifecycle is present. Value *MUST* be one of:

Status	Meaning
draft	Under development; not ready for production use
active	Operational and available for use
deprecated	Superseded; discouraged for new use; may be removed
retired	End-of-life; no longer operational

Table 3

When lifecycle is omitted, no lifecycle assertion is made. Implementations *MUST NOT* assume a default status.

Runtimes *SHOULD* check lifecycle.status before provisioning agents. Runtimes *SHOULD NOT* provision agents with status draft in production environments. Runtimes *SHOULD* warn users when provisioning agents with status deprecated. Runtimes *MUST NOT* provision or execute agents with status retired.

> *Note:* "Provision" and "execute" refer to instantiating an agent for operation. Reading, parsing, validating, analyzing, or migrating from an agent definition is unrestricted regardless of lifecycle status.

5.6.2. effective_date

When present, value *MUST* be a valid ISO 8601 string with timezone. Indicates when the current status took effect.

5.6.3. sunset_date

When present, value *MUST* be a valid ISO 8601 string with timezone. Indicates when the agent will be or was retired. Implementations *SHOULD* warn when sunset_date is in the future and within 30 days. When sunset_date is in the past and status is deprecated, runtimes *SHOULD* treat the agent as retired.

5.6.4. successor

When present, value *MUST* be a string; *SHOULD* be a URI or URN identifying the replacement agent (see Section 6.1 for identifier formats). *SHOULD* be present when status is deprecated or retired. Implementations *SHOULD* warn if successor is present when status is active or draft.

Example:

json { "lifecycle": { "status": "deprecated", "effective_date": "2026-01-15T00:00:00Z", "sunset_date": "2026-08-01T00:00:00Z", "successor": "https://acme.example.com/agents/research-assistant" } }

6. Agent Identity

6.1. Id

Unique identifier for the agent. *OPTIONAL.* When present, value *MUST* be a string and *MUST* be a valid URI [RFC3986] or URN [RFC8141].

Identifier formats, in order of preference:

1. *HTTPS URI (RECOMMENDED):* https://{domain}/agents/{name} -- Provides ownership verification via TLS, direct resolution to the agent's ADL document, and natural integration with .well-known discovery (Section 6.4). The domain authority *SHOULD* serve the ADL document at the identifier URL with media type application/adl+json.

2. *Decentralized Identifier:* did:web:{domain}:agents:{name} -- Provides cryptographic identity binding via the DID Document. Resolution follows the did:web method specification [W3C.DID]. *RECOMMENDED* when cryptographic verification of agent identity is required independent of transport.

3. *URN (offline/catalog use):* urn:adl:{namespace}:{name}:{version} -- Location-independent identifier suitable for air-gapped environments, offline catalogs, and internal registries where network resolution is unavailable. URN identifiers provide naming only; they do not support ownership verification or discovery without an external resolver.

When an agent has both a resolvable identifier (HTTPS URI or DID) and a URN, the resolvable identifier *SHOULD* be used as the primary id value. The URN *MAY* be recorded in metadata for catalog interoperability.

> *Note:* The urn:adl: namespace identifier is used as a convention in this specification but is not yet a registered URN namespace per [RFC8141]. Formal registration with IANA will be pursued in a future revision. Implementations *SHOULD NOT* assume that urn:adl: URNs are globally resolvable.

6.2. Provider

Identifies the organization or entity that provides the agent. *OPTIONAL.* When present, value *MUST* be an object:

Member	Type	Required	Description
name	string	REQUIRED	Provider name
url	string	OPTIONAL	Provider website
contact	string	OPTIONAL	Contact email

Table 4

6.3. Cryptographic Identity

Cryptographic identification for the agent. *OPTIONAL.* When present, value *MUST* be an object:

Member	Type	Required	Description
did	string	OPTIONAL	Decentralized Identifier [W3C.DID]
public_key	object	OPTIONAL	Public key for signature verification

Table 5

At least one of did or public_key *SHOULD* be present. The public_key object, when present, *MUST* contain algorithm (string, REQUIRED) and value (string, Base64-encoded, REQUIRED). Implementations *SHOULD* reject weak algorithms (e.g., RSA below 2048 bits, DSA, ECDSA below P-256). EdDSA (Ed25519, Ed448) is *RECOMMENDED*.

Example (agent identity with DID and public key):

json { "id": "https://acme.example.com/agents/invoice-processor", "provider": { "name": "Acme Corp", "url": "https://acme.example.com", "contact": "ai-platform@acme.example.com" }, "cryptographic_identity": { "did": "did:web:acme.example.com:agents:invoice-processor", "public_key": { "algorithm": "Ed25519", "value": "MCowBQYDK2VwAyEAGb9ECWmEzf6FQbrBZ9w7lshQhqowtrbLDFw4rXAxZuE=" } } }

6.4. Discovery

Agent discovery enables clients to locate agents published by a domain without prior knowledge of individual agent identifiers. Domains hosting ADL agents *MAY* publish a discovery document at the well-known URI [RFC8615]:

https://{domain}/.well-known/adl-agents

The discovery document, when present, *MUST* be a JSON object served with media type application/json and *MUST* contain an agents array. Each entry in the array *MUST* be an object with at least id (string, the agent's identifier per Section 6.1) and adl_document (string, URL to the full ADL document). Entries *MAY* include name, version, description, and status.

Example discovery document:

json { "adl_discovery": "1.0", "agents": [{ "id": "https://acme.example.com/agents/invoice-processor", "adl_document": "https://acme.example.com/agents/invoice-processor/adl.json", "name": "Invoice Processor", "version": "2.0.0", "status": "active" }, { "id": "https://acme.example.com/agents/research-assistant", "adl_document": "https://acme.example.com/agents/research-assistant/adl.json", "name": "Research Assistant", "version": "2.1.0", "status": "active" }] }

Clients performing discovery *MUST* fetch the discovery document over HTTPS. Clients *SHOULD* validate the TLS certificate chain. The discovery document *SHOULD* be cacheable; servers *SHOULD* set appropriate Cache-Control headers.

> *Note:* Registration of .well-known/adl-agents with IANA per [RFC8615] will be pursued alongside the IETF Internet-Draft submission.

7. Model Configuration

7.1. Model

AI model configuration. *OPTIONAL.* When omitted, the runtime determines the model. When present, value *MUST* be an object:

Member	Type	Required	Description
provider	string	OPTIONAL	Model provider identifier
name	string	OPTIONAL	Model identifier
version	string	OPTIONAL	Model version
context_window	number	OPTIONAL	Max context window (tokens)
temperature	number	OPTIONAL	Sampling temperature (0.0-2.0)
max_tokens	number	OPTIONAL	Max output tokens
capabilities	array	OPTIONAL	Required model capabilities

Table 6

capabilities values may include: function_calling, vision, code_execution, streaming.

7.2. System Prompt

System prompt for the agent. *OPTIONAL.* Value *MUST* be a string or an object. When an object, it *MUST* contain template (string, REQUIRED) and *MAY* contain variables (object).

7.2.1. Template Variable Syntax

Variables in templates use the {{variable_name}} syntax and *MUST* conform to the template-var production in Appendix D. Variable names *MUST* begin with a letter (A-Z or a-z) and *MAY* contain letters, digits, and underscores.

Escaping: To include a literal {{ in template text without triggering variable substitution, implementations *MUST* support the escape sequence \{{. A \{{ in the template string is rendered as {{ and is not treated as a variable reference.

Undefined variables: When a template references a variable name not present in variables, the implementation *MUST* treat this as an error (error code ADL-1006) and *MUST NOT* silently substitute an empty string. Implementations *SHOULD* include the undefined variable name in the error detail.

Example:

```
json { "model": { "provider": "acme-ai", "name": "acme-large-2024",
"context_window": 200000, "temperature": 0.7, "max_tokens": 4096,
"capabilities": ["function_calling", "vision"] }, "system_prompt": {
"template": "You are a helpful assistant for {{company_name}}. Today
is {{current_date}}.", "variables": { "company_name": "Acme Corp",
"current_date": "2026-02-18" } } }
```

8. Capabilities

8.1. Tools

Array of tool objects (functions the agent can invoke). *OPTIONAL.* Each tool *MUST* contain name (string, REQUIRED) and description (string, REQUIRED). Each tool *MAY* contain: parameters (JSON Schema), returns (JSON Schema), examples, requires_confirmation (bool), idempotent (bool), read_only (bool), annotations, data_classification (Section 10.4). Tool names *MUST* be unique, *MUST* match ^[a-z][a-z0-9_]*$, and *MUST* conform to the tool-name production in Appendix D. The parameters and returns objects, when present, *MUST* be valid JSON Schema.

The examples member, when present, *MUST* be an array of example objects. Each example object *MAY* contain:

Member	Type	Required	Description
name	string	OPTIONAL	Human-readable name for the example
input	object	OPTIONAL	Example input parameters
output	any	OPTIONAL	Expected output value

Table 7

The annotations member, when present, *MUST* be an object containing implementation hints and metadata. Annotations is an open object -- implementations *MAY* add custom keys. Standard annotation members include:

Member	Type	Required	Description
openapi_ref	string	OPTIONAL	URI to an OpenAPI specification
operation_id	string	OPTIONAL	OpenAPI operation identifier

Table 8

See Section 15.3 for OpenAPI integration details. Implementations *MUST* preserve all annotation members when processing, including unrecognized keys.

Example:

json { "tools": [{ "name": "search_invoices", "description": "Search for invoices by vendor name, date range, or amount.", "parameters": { "type": "object", "properties": { "vendor": { "type": "string", "description": "Vendor name to search" }, "date_from": { "type": "string", "format": "date" }, "date_to": { "type": "string", "format": "date" } }, "required": [] }, "returns": { "type": "array", "items": { "type": "object" } }, "examples": [{ "name": "Search by vendor", "input": { "vendor": "Acme Supplies" }, "output": [{ "id": "INV-001", "amount": 1500.00 }] }], "idempotent": true, "read_only": true, "annotations": { "openapi_ref": "https://api.acme.example.com/openapi.json", "operation_id": "searchInvoices" }, "data_classification": { "sensitivity": "confidential" } }] }

8.2. Resources

Array of resource objects (data sources the agent can access). *OPTIONAL.* Each resource *MUST* contain name (string, REQUIRED) and type (string, REQUIRED). type *MUST* be one of: vector_store, knowledge_base, file, api, database. Each resource *MAY* contain: description, uri, mime_types, schema, annotations, data_classification (Section 10.4). Resource names *MUST* be unique.

The mime_types member, when present, *MUST* be an array of strings. Each value *MUST* be a valid MIME type (e.g., "application/json", "text/plain").

The schema member, when present, *MUST* be a valid JSON Schema describing the structure of the resource’s data.

The annotations member, when present, *MUST* be an object. Same semantics as tool.annotations -- an open object for implementation hints that *MUST* be preserved when processing.

Example:

json { "resources": [{ "name": "invoice_store", "type": "vector_store", "description": "Vector store containing indexed invoice documents for semantic search.", "uri": "https://store.acme.example.com/invoices", "mime_types": ["application/pdf", "application/json"], "data_classification": { "sensitivity": "confidential" } }] }

8.3. Prompts

Array of prompt objects (reusable prompt templates). *OPTIONAL.* Each prompt *MUST* contain name (string, REQUIRED) and template (string, REQUIRED). Each prompt *MAY* contain description, arguments (JSON Schema). Template arguments use {{argument_name}} and *MUST* conform to the template-var production in Appendix D. Prompt names *MUST* be unique.

Example:

json { "prompts": [{ "name": "summarize_invoice", "description": "Summarizes an invoice for a reviewer.", "template": "Summarize the following invoice for {{reviewer_role}}:\n\n{{invoice_text}}\n\nHighlight amounts over {{threshold}}.", "arguments": { "type": "object", "properties": { "reviewer_role": { "type": "string" }, "invoice_text": { "type": "string" }, "threshold": { "type": "number" } }, "required": ["reviewer_role", "invoice_text"] } }] }

9. Permissions

The permissions member defines the agent’s operational boundaries. *OPTIONAL.* When present, value *MUST* be an object containing one or more permission domain members.

9.1. Permissions Model

Domain	Description
network	Network access boundaries
filesystem	Filesystem access boundaries

environment	Environment variable access
execution	Process execution boundaries
resource_limits	Resource consumption limits

Table 9

Permissions operate on a *deny-by-default* model. Runtimes *MUST* deny any capability not explicitly granted in the permissions member. Runtimes *MUST* enforce declared permissions. Runtimes that cannot enforce a specific permission domain *MUST* warn users before execution and *SHOULD* refuse to execute the agent unless the user explicitly acknowledges the limitation.

When the permissions member is omitted from an ADL document, no permissions are granted to the agent. Runtimes *MUST* treat the absence of permissions as equivalent to an empty permissions object -- the agent has no granted capabilities.

When a specific permission domain (e.g., network, filesystem) is omitted from the permissions object, all operations in that domain are denied. For example, if permissions is present but does not contain network, the agent *MUST* have no network access.

Runtimes *MUST NOT* infer, assume, or provide default permissions when permissions or a permission domain is absent.

9.1.1. Conflict Resolution

When a value matches both an allowed_* pattern and a denied_* pattern within the same permission domain, the denied_* pattern *MUST* take precedence. The agent *MUST NOT* be granted access to any value matched by a denied_* pattern, regardless of whether it also matches an allowed_* pattern. This deny-takes-precedence rule ensures that explicit exclusions cannot be overridden by broad allow patterns.

Example: If allowed_variables is ["APP_*"] and denied_variables is ["APP_SECRET_*"], the variable APP_SECRET_KEY is *denied* even though it matches APP_*.

9.2. Network

May contain: allowed_hosts (array of host patterns), allowed_ports, allowed_protocols, deny_private (bool). Host patterns support exact match and *.example.com.

Host patterns in allowed_hosts *MUST* conform to the pattern syntax defined in Section 4.4.

9.3. Filesystem

May contain: allowed_paths (array of { path, access } where access is read, write, or read_write), denied_paths.

Path patterns in allowed_paths[*].path and denied_paths *MUST* conform to the pattern syntax defined in Section 4.4. The ** multi-segment wildcard is valid in filesystem path patterns.

9.4. Environment

May contain: allowed_variables, denied_variables (patterns with wildcards, e.g., APP_*).

Variable patterns in allowed_variables and denied_variables *MUST* conform to the pattern syntax defined in Section 4.4.

9.5. Execution

May contain: allowed_commands, denied_commands, allow_shell (bool).

Command patterns in allowed_commands and denied_commands *MUST* conform to the pattern syntax defined in Section 4.4.

9.6. Resource Limits

May contain: max_memory_mb, max_cpu_percent, max_duration_sec, max_concurrent.

Example (complete permissions object):

json { "permissions": { "network": { "allowed_hosts": ["api.acme.example.com", "*.storage.example.com"], "allowed_ports": [443], "allowed_protocols": ["https"], "deny_private": true }, "filesystem": { "allowed_paths": [{ "path": "/data/invoices/**", "access": "read" }, { "path": "/tmp/processing/**", "access": "read_write" }], "denied_paths": ["/tmp/processing/**/secrets"] }, "environment": { "allowed_variables": ["APP_*", "INVOICE_*"], "denied_variables": ["APP_SECRET_*"] }, "execution": { "allowed_commands": ["python3", "jq"], "allow_shell": false }, "resource_limits": { "max_memory_mb": 512, "max_cpu_percent": 25, "max_duration_sec": 300 } } }

10. Security

The security member defines security requirements. *OPTIONAL.* When present, value *MUST* be an object that *MAY* contain authentication, encryption, and attestation.

10.1. Authentication

May contain: type (one of none, api_key, oauth2, oidc, mtls), required (bool). Type-specific members (e.g., OAuth2: scopes, token_endpoint; OIDC: issuer, audience) *MAY* be present.

10.2. Encryption

May contain: in_transit (required, min_version), at_rest (required, algorithm).

10.3. Attestation

May contain: type (one of self, third_party, verifiable_credential), issuer, issued_at, expires_at (ISO 8601), signature (object). Implementations *SHOULD* warn when expires_at is in the past or within 30 days.

Signature object: When present, *MUST* contain algorithm, value (Base64url-encoded), signed_content ("canonical" or "digest"). When signed_content is "digest", *MUST* also include digest_algorithm and digest_value. Supported algorithms include Ed25519 (RECOMMENDED), Ed448, ES256/384/512, RS256, PS256 (RSA >= 2048). Verification: remove signature, serialize with JCS [RFC8785], verify digest if applicable, resolve public key from cryptographic_identity, verify signature.

Example:

json { "security": { "authentication": { "type": "oauth2", "required": true, "scopes": ["invoices:read", "invoices:write"], "token_endpoint": "https://auth.acme.example.com/oauth/token" }, "encryption": { "in_transit": { "required": true, "min_version": "TLS1.3" }, "at_rest": { "required": true, "algorithm": "AES-256-GCM" } }, "attestation": { "type": "third_party", "issuer": "https://trust.acme.example.com", "issued_at": "2026-01-01T00:00:00Z", "expires_at": "2027-01-01T00:00:00Z" } } }

10.4. Data Classification

The data_classification member declares the sensitivity and categories of data the agent may access, process, or produce. *REQUIRED.* Value *MUST* be an object.

Data classification is required by NIST FIPS 199, NIST SP 800-60, ISO 27001:2022 Annex A.5.12, FedRAMP, SOC 2, and CMMC. It is the foundational step of security categorization across all major compliance frameworks.

This member is a *reusable composable attribute*. In addition to the required top-level declaration, it *MAY* also appear within individual tools[*] or resources[*] objects to classify specific capabilities. When present on both the top level and a tool or resource, the tool/resource-level classification applies to that capability.

10.4.1. High-Water Mark Rule

The top-level data_classification.sensitivity *MUST* be greater than or equal to the highest sensitivity value declared in any tool-level or resource-level data_classification within the same document. This follows the FIPS 199 high-water mark principle: a system’s overall security categorization is the highest value among its constituent information types.

The sensitivity ordering from lowest to highest is: public < internal < confidential < restricted.

Sensitivity levels align with NIST FIPS 199 impact categorization and ISO 27001:2022 Annex A.5.12 information classification.

Member	Type	Required	Description
sensitivity	string	REQUIRED	Information sensitivity level
categories	array	OPTIONAL	Broad information categories handled
retention	object	OPTIONAL	Data retention requirements
handling	object	OPTIONAL	Data handling constraints

Table 10

10.4.2. sensitivity

REQUIRED when data_classification is present. Value *MUST* be one of:

Value	Definition
public	Information approved for unrestricted disclosure
internal	Information limited to organizational use
confidential	Information requiring protection; unauthorized disclosure could cause harm
restricted	Information requiring the highest level of protection; unauthorized disclosure could cause severe harm

Table 11

10.4.3. categories

When present, *MUST* be a non-empty array. Each item *MUST* be one of:

Value	Definition
pii	Personally Identifiable Information
phi	Protected Health Information (HIPAA)
financial	Financial data (PCI-DSS, GLBA, SOX scope)
credentials	Authentication credentials, secrets, keys
intellectual_property	Trade secrets, proprietary algorithms, business-sensitive data
regulatory	Data subject to specific regulatory requirements

Table 12

Profiles *MAY* define additional category values.

10.4.4. retention

When present, *MUST* be an object. *MAY* contain:

Member	Type	Description
min_days	number	Minimum retention period in days
max_days	number	Maximum retention period in days
policy_uri	string	URI to the governing retention policy

Table 13

When both min_days and max_days are present, min_days *MUST* be less than or equal to max_days.

10.4.5. handling

When present, *MUST* be an object. *MAY* contain:

Member	Type	Description
encryption_required	bool	Whether data must be encrypted at rest
anonymization_required	bool	Whether data must be anonymized before processing
cross_border_restricted	bool	Whether data may not leave jurisdictional boundaries
logging_required	bool	Whether all access must be logged

Table 14

10.4.6. Profile Extensions

Profiles *MAY* add domain-specific sub-objects within data_classification to provide granular classification vocabularies. For example, a healthcare profile may add a healthcare sub-object with PHI type enumerations, and a financial profile may add a financial sub-object with financial data type enumerations. Multiple profile extensions compose naturally within the same data_classification object. See Section 13 for profile composition rules.

Example (top-level and tool-level data classification demonstrating the high-water mark rule):

json { "data_classification": { "sensitivity": "confidential", "categories": ["financial", "pii"], "retention": { "max_days": 2555, "policy_uri": "https://acme.example.com/data-retention" }, "handling": { "encryption_required": true, "logging_required": true } }, "tools": [{ "name": "get_invoice_details", "description": "Returns detailed invoice data including PII.", "data_classification": { "sensitivity": "confidential", "categories": ["financial", "pii"] } }, { "name": "get_invoice_summary", "description": "Returns anonymized invoice summary.", "data_classification": { "sensitivity": "internal" } }] }

The top-level sensitivity of "confidential" satisfies the high-water mark rule: it equals the highest tool-level value ("confidential" for get_invoice_details).

11. Runtime Behavior

The runtime member configures agent runtime behavior. *OPTIONAL.* When present, value *MUST* be an object.

11.1. Input Handling

May contain: max_input_length, content_types, sanitization.

The sanitization member, when present, *MUST* be an object describing input sanitization rules. It *MAY* contain:

Member	Type	Required	Description
enabled	boolean	OPTIONAL	Whether input sanitization is active
strip_html	boolean	OPTIONAL	Whether to strip HTML tags from input
max_input_length	number	OPTIONAL	Maximum input length in characters

Table 15

The content_types member, when present, *MUST* be an array of strings. Each value *MUST* be a valid MIME type specifying an accepted input content type.

11.2. Output Handling

May contain: max_output_length, format, streaming (bool).

The format member, when present, *MUST* be a string specifying the default output format. Value *MUST* be one of: "text", "json", "markdown", "html".

11.3. Tool Invocation

May contain: parallel (bool), max_concurrent, timeout_ms, retry_policy.

The retry_policy member, when present, *MUST* be an object describing retry behavior for tool invocations. It *MAY* contain:

Member	Type	Required	Description
max_retries	number	OPTIONAL	Maximum number of retry attempts
backoff_strategy	string	OPTIONAL	One of: "fixed", "exponential", "linear"
initial_delay_ms	number	OPTIONAL	Initial delay between retries in milliseconds
max_delay_ms	number	OPTIONAL	Maximum delay between retries in milliseconds

Table 16

11.4. Error Handling

May contain: on_tool_error (abort, continue, or retry), max_retries, fallback_behavior.

The fallback_behavior member, when present, *MUST* be an object describing behavior when errors occur and on_tool_error does not resolve the situation. It *MAY* contain:

Member	Type	Required	Description
action	string	OPTIONAL	One of: "return_error", "use_default", "skip"
default	any	OPTIONAL	Default value to return when action is "use_default"
message	string	OPTIONAL	User-facing message on fallback

Table 17

Example:

json { "runtime": { "input_handling": { "max_input_length": 32768, "content_types": ["text/plain", "application/json"], "sanitization": { "enabled": true, "strip_html": true } }, "output_handling": { "format": "json", "max_output_length": 8192, "streaming": false },

"tool_invocation": { "parallel": true, "max_concurrent": 3, "timeout_ms": 30000, "retry_policy": { "max_retries": 2, "backoff_strategy": "exponential", "initial_delay_ms": 500, "max_delay_ms": 5000 } }, "error_handling": { "on_tool_error": "retry", "max_retries": 2, "fallback_behavior": { "action": "return_error", "message": "Invoice processing temporarily unavailable." } } } }

12. Metadata

The metadata member provides additional information. *OPTIONAL.* When present, value *MUST* be an object.

12.1. Authors

Array of author objects. Each *MAY* contain name, email, url.

12.2. License

String: SPDX license identifier or URI to license document.

12.3. Documentation

String: URI to documentation.

12.4. Repository

String: URI to source repository.

12.5. Tags

Array of strings. *SHOULD* be lowercase, alphanumeric and hyphens only. Tags *SHOULD* conform to the tag production in Appendix D.

12.6. Example

json { "metadata": { "authors": [{ "name": "Platform Team", "email": "platform@example.com", "url": "https://example.com/team/platform" }], "license": "Apache-2.0", "documentation": "https://docs.example.com/agents/invoice-processor", "repository": "https://github.com/example/invoice-processor", "tags": ["finance", "invoice", "production"] } }

13. Profiles

The profiles member declares which profiles the document conforms to. *OPTIONAL.* Value *MUST* be an array of profile identifiers (URIs or registered names). When a profile is declared: the document *MUST* satisfy all profile requirements, *MAY* use profile-defined members, and validators *SHOULD* check profile-specific rules.

ADL defines two categories of profiles:

* *Standard profiles* define domain-specific top-level members and validation rules. Standard profiles use the urn:adl:profile:* namespace and *SHOULD* be registered with the IANA profile registry (Section 13.5) to prevent naming conflicts. Examples: Governance (urn:adl:profile:governance:1.0), Healthcare, Financial.

* *Vendor profiles* declare vendor-specific extensions with schema validation, targeting the extensions namespace rather than defining new top-level members. Vendor profiles use URI identifiers controlled by the vendor (e.g., https://acme.com/adl/extensions/v1) and do not require registration -- the reverse-domain namespace provides collision prevention through DNS ownership. See Section 13.4.

Both categories use the same allOf composition mechanism (Section 13.1) and *MAY* appear together in a document’s profiles array.

13.1. Profile Schema Composition

Profiles extend the base ADL schema using the JSON Schema 2020-12 allOf composition mechanism. Each profile publishes a JSON Schema that:

1. References the base ADL schema via allOf with $ref.

2. Declares the profile’s additional top-level members in its own properties.

3. Adds unevaluatedProperties: false to close the composed schema, ensuring only base ADL members, profile-defined members, and extensions members are accepted.

The base ADL schema (Appendix A) does not restrict unknown top-level properties -- it declares properties and patternProperties but omits additionalProperties and unevaluatedProperties. This allows profile schemas to add members via composition without conflict. For

documents that do not declare any profiles, validators *SHOULD* use the strict schema (schema-strict.json), which adds unevaluatedProperties: false to reject unknown top-level members.

Profile schemas *MUST NOT* redefine core ADL members with incompatible types. Profiles *MAY*:

* Add top-level members.

* Add members to existing objects (e.g., extending data_classification with domain-specific sub-objects).

* Define validation rules.

* Require specific values for optional core members.

* Use conditional validation (if/then) to enforce tier-based or context-dependent requirements.

13.2. Multi-Profile Composition

When a document declares multiple profiles, the document *MUST* satisfy all declared profile requirements. Validators compose profile schemas using allOf -- each profile's schema is included as an element. JSON Schema allOf uses "strictest wins" semantics: if any profile requires a member, the composed result requires it.

Profiles *MUST* be designed for independent composition. A profile's validation rules *MUST NOT* assume the absence of members defined by other profiles. For standard profiles, the IANA profile registry designated expert review (see Section 13.5) prevents cross-profile field naming conflicts. Vendor profiles avoid conflicts through their reverse-domain namespace isolation.

13.3. Profile Dependencies

A profile *MAY* declare dependencies on other profiles. When a profile declares a dependency, documents using that profile *MUST* also satisfy the dependency profile's requirements. The profiles array *MUST* include all transitive dependencies.

At the schema level, a dependent profile composes its parent via allOf:

```
json { "allOf": [ { "$ref": "https://adl-spec.org/0.2/schema.json" },
{ "$ref": "https://adl-spec.org/profiles/governance/1.0/schema.json"
} ], "properties": { "hipaa_data_handling": { "type": "object" } },
"unevaluatedProperties": false }
```

A dependent profile *MAY* tighten constraints from its parent (e.g., make an optional parent field required, narrow an enum). A dependent profile *MUST NOT* loosen constraints from its parent (e.g., make a required parent field optional). This follows from allOf semantics -- the parent's constraints remain in force.

If a dependent profile needs a parent field to not be required, this indicates a design issue. Resolutions include: refactoring the parent into a base profile with looser constraints, changing the relationship from dependency to sibling, or revising the parent profile in a new major version.

13.4. Vendor Profiles

A *vendor profile* is a profile published by an organization to declare vendor-specific extensions with schema validation. Vendor profiles use the same allOf composition mechanism as standard profiles (Section 13.1) but target the extensions namespace rather than defining new top-level members. See Section 13 for an overview of the standard/vendor profile taxonomy.

Vendor profiles use URI identifiers controlled by the vendor (e.g., https://acme.com/adl/extensions/v1). The urn:adl:profile:* namespace is reserved for standard profiles. Vendor profiles *MUST NOT* use this namespace.

A vendor profile *MAY* add schema constraints to the extensions object at any level, validating that its reverse-domain namespace contains the expected structure. The profile schema references the base ADL schema via allOf and declares properties for extensions within the relevant objects.

A vendor profile *MAY* declare a dependency on a standard profile and add schema constraints to extensions within that profile's objects. The vendor profile composes its dependency via allOf and adds extensions constraints inside the profile-defined objects. This enables vendors to extend profile-defined objects without redefining them.

Vendor profiles are subject to the following constraints:

* Vendor profiles *MUST NOT* redefine core ADL members or standard profile members with incompatible types.

* Vendor profiles *MUST* only add schema constraints within their own reverse-domain namespace under extensions.

* A vendor profile's extensions schema applies only when the vendor profile is declared in the document's profiles array.

* Documents *MAY* include extensions data for a vendor without declaring the vendor's profile. In this case, the data is preserved but unvalidated -- implementations treat it as opaque.

* Multiple vendor profiles compose independently. Each vendor's extensions constraints apply only within its own namespace.

Vendor profiles do not require IANA registration. The reverse-domain namespace provides collision prevention through DNS ownership.

Vendors *SHOULD*:

* Publish their profile schema at a stable, dereferenceable URI.

* Version their profile schemas (e.g., /v1/, /v2/).

* Document the semantics of their extension fields.

13.5. Standard Profile Registration

Standard profile identifiers *SHOULD* be registered to prevent naming conflicts. Only standard profiles -- those using the urn:adl:profile:* namespace -- are subject to registration. Vendor profiles rely on reverse-domain namespace isolation and do not require registration (see Section 13.4).

The registration authority (e.g., IANA profile registry) *MUST* employ designated expert review to ensure:

1. New standard profiles do not redefine members from existing profiles with incompatible semantics.

2. New standard profiles do not introduce field names that conflict with existing profiles.

3. Dependencies between profiles are explicitly declared and acyclic.

If a member becomes cross-cutting (needed by multiple standard profiles), the registration authority *MAY* recommend promoting it to the core ADL specification.

13.6. Example

```
json { "adl_spec": "0.2.0", "name": "Invoice Processor", "version":
"2.0.0", "description": "Processes invoices with governance and
financial compliance.", "data_classification": { "sensitivity":
"confidential", "categories": ["financial"] }, "profiles": [
"urn:adl:profile:governance:1.0", "urn:adl:profile:financial:1.0" ] }
```

14. Processing ADL Documents

14.1. Parsing

Implementations *MUST* parse ADL as JSON [RFC8259], *MUST* reject invalid JSON, and *MUST* reject documents where the top-level value is not a JSON object.

14.2. Validation

Implementations *MUST* validate ADL documents against the JSON Schema defined in Appendix A. Implementations *MUST* validate the following semantic rules:

Rule	Description
VAL-01	adl_spec MUST match a supported version
VAL-02	Tool names MUST be unique
VAL-03	Resource names MUST be unique
VAL-04	Prompt names MUST be unique
VAL-05	Timestamps MUST be valid ISO 8601
VAL-06	URIs MUST be valid per RFC 3986
VAL-07	JSON Schema in parameters/returns MUST be valid
VAL-08	Profile requirements MUST be satisfied
VAL-09	lifecycle.status MUST be a valid status value if present
VAL-10	lifecycle.effective_date MUST be valid ISO 8601 if present
VAL-11	lifecycle.sunset_date MUST be valid ISO 8601 if

	present
VAL-12	lifecycle.successor MUST be a valid URI if present
VAL-13	Tool names MUST match ^[a-z][a-z0-9_]*$
VAL-14	Resource type MUST be a valid resource type value
VAL-15	model.temperature MUST be between 0.0 and 2.0 if present
VAL-16	security.authentication.type MUST be a valid authentication type if present
VAL-17	security.attestation.type MUST be a valid attestation type if present
VAL-18	runtime.error_handling.on_tool_error MUST be a valid error action if present
VAL-19	runtime.output_handling.format MUST be a valid format value if present
VAL-20	model.capabilities items MUST be valid capability values if present
VAL-21	Host patterns MUST conform to Section 4.4 pattern syntax
VAL-22	Filesystem path patterns MUST conform to Section 4.4 pattern syntax
VAL-23	Environment variable patterns MUST conform to Section 4.4 pattern syntax
VAL-24	Attestation signature.signed_content value "digest" MUST have digest_algorithm and digest_value present
VAL-25	data_classification.sensitivity MUST be a valid sensitivity level if present
VAL-26	data_classification.categories items MUST be valid category values if present
VAL-27	data_classification.retention.min_days MUST be less than or equal to max_days when both are present
VAL-28	Top-level data_classification.sensitivity MUST be

	>= the highest sensitivity in any tool or resource data_classification (high-water mark)	

Table 18

Implementations *MAY* perform additional validation based on declared profiles.

14.3. Unknown Members

Implementations *MUST* preserve unrecognized members when round-tripping. Implementations *MUST NOT* reject documents containing extensions with unknown vendor namespaces. Implementations *MAY* warn on unknown non-extension, non-profile members.

15. Interoperability

15.1. A2A Agent Card Generation

Implementations *SHOULD* support generating A2A Agent Cards from ADL (e.g., name, description, version, tools->skills, cryptographic_identity.did->id, security.authentication->authentication).

15.2. MCP Server Configuration

Implementations *SHOULD* support generating MCP server configurations (name, description, version, tools, resources, prompts).

15.3. OpenAPI Integration

Tools that invoke HTTP APIs *MAY* reference OpenAPI specs. The tool annotations object *MAY* contain openapi_ref (URI) and operation_id.

16. Errors

16.1. Error Format

Implementations *SHOULD* return errors in a consistent format, e.g.:

json { "errors": [{ "code": "ADL-1001", "title": "Invalid JSON", "detail": "Unexpected token at line 42, column 15", "source": { "pointer": "/tools/0/name" } }] }

The source object *MAY* contain: pointer (JSON Pointer to the error location), line (1-indexed), column (1-indexed).

16.2. Error Codes

Code	Category	Description
ADL-1001	Parse	Invalid JSON syntax
ADL-1002	Parse	Document is not a JSON object
ADL-1003	Schema	Missing required member
ADL-1004	Schema	Invalid member type
ADL-1005	Schema	Invalid enum value
ADL-1006	Schema	Value does not match pattern
ADL-2001	Semantic	Unsupported ADL version
ADL-2002	Semantic	Duplicate tool name
ADL-2003	Semantic	Duplicate resource name
ADL-2004	Semantic	Duplicate prompt name
ADL-2005	Semantic	Invalid timestamp format
ADL-2006	Semantic	Invalid URI format
ADL-2007	Semantic	Invalid JSON Schema
ADL-2008	Semantic	Invalid tool name pattern
ADL-2009	Semantic	Invalid resource type value
ADL-2010	Semantic	Temperature out of range
ADL-2011	Semantic	Invalid authentication type
ADL-2012	Semantic	Invalid attestation type
ADL-2013	Semantic	Invalid error handling action
ADL-2014	Semantic	Invalid output format
ADL-2015	Semantic	Invalid model capability
ADL-2016	Semantic	Invalid host pattern syntax

ADL-2017	Semantic	Invalid filesystem path pattern
ADL-2018	Semantic	Invalid environment variable pattern
ADL-2019	Semantic	Missing digest fields for digest-mode signature
ADL-2020	Semantic	Invalid data classification sensitivity level
ADL-2021	Semantic	Invalid data classification category
ADL-2022	Semantic	Retention min_days exceeds max_days
ADL-2023	Semantic	Top-level sensitivity below tool/resource sensitivity (high-water mark violation)
ADL-3001	Profile	Profile requirements not satisfied
ADL-3002	Profile	Unknown profile
ADL-4001	Security	Weak key algorithm
ADL-4002	Security	Invalid signature
ADL-4003	Security	Expired attestation
ADL-5001	Lifecycle	Invalid lifecycle status value
ADL-5002	Lifecycle	Successor present on active/draft agent
ADL-5003	Lifecycle	Sunset date in the past with non-retired status

Table 19

16.3. Error Source Examples

The source.pointer member uses JSON Pointer [RFC6901] to identify the location of the error within the ADL document. The following examples illustrate source values for representative error codes from each category:

```
json // ADL-1003 (Schema): Missing required member
"data_classification" { "code": "ADL-1003", "title": "Missing
required member", "detail": "Required member 'data_classification' is
missing", "source": { "pointer": "" } }

json // ADL-2002 (Semantic): Duplicate tool name at index 2 { "code":
"ADL-2002", "title": "Duplicate tool name", "detail": "Tool name
'search_documents' already defined at index 0", "source": {
"pointer": "/tools/2/name" } }

json // ADL-2016 (Semantic): Invalid host pattern in permissions {
"code": "ADL-2016", "title": "Invalid host pattern syntax", "detail":
"Pattern '**' is not a valid host pattern", "source": { "pointer":
"/permissions/network/allowed_hosts/1" } }

json // ADL-2023 (Semantic): High-water mark violation on a tool {
"code": "ADL-2023", "title": "High-water mark violation", "detail":
"Tool 'query_records' has sensitivity 'confidential' which exceeds
top-level 'internal'", "source": { "pointer":
"/tools/1/data_classification/sensitivity" } }

json // ADL-3001 (Profile): Profile requirement not satisfied {
"code": "ADL-3001", "title": "Profile requirements not satisfied",
"detail": "Governance profile requires 'compliance' member",
"source": { "pointer": "/profiles/0" } }

json // ADL-4001 (Security): Weak key algorithm { "code": "ADL-4001",
"title": "Weak key algorithm", "detail": "Algorithm 'RS256' with
1024-bit key does not meet minimum strength requirements", "source":
{ "pointer": "/security/attestation/public_key" } }

json // ADL-5002 (Lifecycle): Successor on active agent { "code":
"ADL-5002", "title": "Successor present on non-retired agent",
"detail": "Member 'successor' is only valid when lifecycle.status is
'retired'", "source": { "pointer": "/lifecycle/successor" } }
```

17. IANA Considerations

17.1. Media Type

This document requests IANA to register the application/adl+json media type in the "Media Types" registry in accordance with [RFC6838].

* *Type name:* application

* *Subtype name:* adl+json

* *Required parameters:* None

* *Optional parameters:*

  - profile -- A comma-separated list of ADL profile identifiers (URIs or registered names from the ADL Profile Registry defined in Section 17.2) that the document conforms to. Each identifier *MUST* be a URI conforming to [RFC3986]. Consumers that do not recognize a profile identifier *MAY* ignore the parameter and *MUST* preserve it when retransmitting the document.

* *Encoding considerations:* binary -- ADL documents are JSON text sequences encoded in UTF-8 [RFC8259]. No other character encoding is permitted. Consistent with [RFC8259], UTF-8 without a byte-order mark (BOM) is *RECOMMENDED*.

* *Security considerations:* ADL documents declare agent behavior including permission grants, system prompt templates, tool invocation configuration, and cryptographic identity. Processors *MUST* treat content from untrusted sources with appropriate caution. Template variables in system_prompt and prompt templates use a {{variable_name}} substitution syntax; processors *MUST* sanitize variable values before substitution to prevent prompt injection attacks that could alter agent behavior. ADL documents include URI references in fields such as $schema, openapi_ref, documentation, and repository; processors *MUST NOT* automatically dereference these URIs from untrusted documents, as doing so may target internal network resources and enable Server-Side Request Forgery (SSRF). Documents that declare broad permissions (e.g., a bare * wildcard in allowed_hosts) represent elevated risk and *SHOULD* require explicit human review before deployment. Processors *SHOULD* impose limits on document size, JSON nesting depth, and array lengths to prevent resource exhaustion from adversarially crafted documents. For a comprehensive treatment of all security considerations applicable to this media type, see Section 18.

* *Interoperability considerations:* ADL documents *MUST* be processed as JSON [RFC8259] regardless of authoring format. YAML is a common authoring convenience, but processors *MUST* operate on the JSON form; documents intended to be signed using JCS [RFC8785] *MUST* be serialized as JSON before signing. Profile declarations -- whether via the profile optional parameter or the profiles document member -- allow multiple profiles to compose within a single document; consumers that partially implement profile requirements *SHOULD* process the members they recognize and preserve unrecognized members per Section 14.3. Validation

against the JSON Schema defined in Appendix A provides a baseline interoperability check. Implementations that generate A2A Agent Cards or MCP server configurations from ADL documents *SHOULD* follow the mappings defined in Section 15. Producers *SHOULD* include the $schema member to enable tooling-assisted validation.

* *Published specification:* [this document]

* *Applications that use this media type:* AI agent platforms, agent registries, development tools, orchestration frameworks, and runtime environments that provision and manage AI agents.

* *Fragment identifier considerations:* Fragment identifiers for resources of this type *SHOULD* be interpreted as JSON Pointer expressions [RFC6901] identifying a location within the ADL document object.

* *Additional information:*

  - Deprecated alias names for this type: N/A

  - Magic number(s): N/A

  - File extension(s): .adl.json, .adl

  - Macintosh file type code(s): N/A

  - Object Identifiers: N/A

* *Person and email address to contact for further information:* See the Author’s Address section of this document.

* *Intended usage:* COMMON

* *Restrictions on usage:* None

* *Author:* See the Author’s Address section of this document.

* *Change controller:* IETF

17.2. Profile Registry

IANA is requested to create and maintain a new registry titled *"ADL Profile Registry"* within a new "Agent Definition Language (ADL)" registry group.

Registration Policy: Specification Required [RFC8126]. The designated expert reviews registration requests to verify that the profile is documented in a publicly available, stable specification and that all required registration template fields are complete.

Registration Template: Parties wishing to register a profile *MUST* provide all of the following fields:

Field	Description
Identifier (URI)	A URI that uniquely identifies the profile, conforming to [RFC3986]. The URI *SHOULD* be dereferenceable and return a human-readable description of the profile.
Name	A short human-readable name for the profile (e.g., "ADL Governance Profile").
Version	The profile version string in MAJOR.MINOR.PATCH semantic versioning format.
Specification Reference	A stable, publicly accessible URI or document reference for the profile specification. The specification *MUST* define all profile-required members, validation rules, and any additional semantics added by the profile.
ADL Version Compatibility	The ADL specification version(s) with which the profile is designed to operate (e.g., "0.1.x").
Contact	Name and email address of the person or group responsible for the profile registration.
Status	One of: active (currently maintained) or deprecated (superseded or abandoned).

Table 20

Initial Registry Contents:

Identifier (URI)	Name	Version
urn:adl:profile:governance:1.0	ADL Governance Profile	1.0.0
urn:adl:profile:portfolio:1.0	ADL Portfolio Profile	1.0.0
urn:adl:profile:healthcare:1.0	ADL Healthcare Profile	1.0.0
urn:adl:profile:financial:1.0	ADL Financial Profile	1.0.0

Table 21

All initial entries reference Appendix C of this document, target ADL compatibility 0.1.x, are active, and list the Author’s Address as contact.

Designated Expert Criteria: The designated expert *SHOULD* evaluate requests against the following criteria:

1. *Publicly available specification:* The profile specification *MUST* be accessible at a stable, public URI. Specifications behind paywalls or access controls are not acceptable for registration.

2. *Non-conflict with core ADL:* The profile *MUST NOT* redefine or contradict normative requirements of the core ADL specification. Profiles *MAY* add new members, constrain optional members to a subset of permitted values, or require that optional core members be present.

3. *Complete registration template:* All required template fields *MUST* be present and non-empty. Incomplete registrations *MUST* be returned to the submitter.

4. *Stable identifier:* The profile URI *SHOULD* be dereferenceable and *SHOULD* remain stable over time. Ephemeral or frequently changing URIs are not acceptable.

5. *Legitimate purpose:* The profile *SHOULD* address a genuine domain or deployment need not already covered by an existing active registered profile.

17.3. URN Namespace

IANA is requested to register the adl URN namespace identifier in the "Formal URN Namespaces" registry in accordance with [RFC8141].

* *Namespace Identifier:* adl
* *Version:* 1
* *Date:* [date of publication]
* *Registrant:* See the Author’s Address section of this document.
* *Purpose:* The urn:adl: namespace provides persistent, location-independent identifiers for ADL agents, profiles, and related artifacts. These identifiers are intended for use in offline catalogs, air-gapped environments, and internal registries where network resolution is unavailable. For connected environments, HTTPS URIs (Section 6.1) are the *RECOMMENDED* identifier format.
* *Syntax:* URNs in this namespace conform to the following structure: urn:adl:{type}:{namespace}:{name}:{version} where {type} is one of agent or profile, {namespace} is a lowercase alphanumeric organization identifier, {name} is a lowercase alphanumeric resource name with hyphens, and {version} is a semantic version string. The formal syntax is defined by the adl-urn production in Appendix D.
* *Assignment:* Sub-namespace assignment under urn:adl:profile: is governed by the ADL Profile Registry (Section 17.2). Sub-namespace assignment under urn:adl:agent: is at the discretion of the namespace holder; no central registry is required for agent URNs.
* *Security and Privacy:* URN identifiers in this namespace are opaque strings and carry no inherent security properties. Implementations *MUST NOT* infer ownership, trust, or authorization from a urn:adl: identifier alone. Verification of agent identity *MUST* rely on the mechanisms described in Section 6.3 (Cryptographic Identity) and Section 10.3 (Attestation). See Section 18 for comprehensive security considerations.

17.4. Well-Known URI

IANA is requested to register the adl-agents well-known URI suffix in the "Well-Known URIs" registry in accordance with [RFC8615].

* *URI suffix:* adl-agents

* *Change controller:* IETF

* *Specification document:* Section 6.4 of [this document]

* *Status:* permanent

* *Related information:* The well-known URI https://{domain}/.well-known/adl-agents returns a JSON document listing all ADL agents published by the domain authority. The document format is defined in Section 6.4. The resource *MUST* be served over HTTPS.

18. Security Considerations

18.1. Document Integrity

ADL documents define agent behavior, permission grants, and security requirements. The trust model for an ADL document depends on its provenance and the integrity mechanisms applied to it. Unsigned ADL documents from untrusted or unverified sources *MUST* be treated as potentially malicious.

When a document includes a cryptographic signature in security.attestation.signature, implementations *MUST* verify the signature before acting on the document's permission or security declarations. Signature verification requires serializing the document (with the signature object removed) using JCS [RFC8785] to produce a canonical byte sequence, then verifying the resulting digest using the algorithm and public key declared in cryptographic_identity. Implementations *MUST* reject documents that claim to be signed but whose signature does not verify. Implementations *SHOULD* warn when processing signed documents whose attestation has expired (expires_at is in the past). An ADL document whose permissions or capabilities have been modified after signing will produce a different canonical byte sequence and fail signature verification; this is the intended behavior and provides protection against privilege escalation via document tampering.

18.2. Sensitive Data in ADL Documents

ADL documents *SHOULD NOT* contain secrets, credentials, or other sensitive data in plaintext. Fields such as system_prompt, provider.contact, metadata.authors, and tool parameter examples may inadvertently expose confidential information if documents are logged, cached, or transmitted without adequate access controls.

API keys, passwords, private keys, bearer tokens, and other authentication material *MUST NOT* appear as literal string values in ADL documents. Where agent configuration requires secret values at runtime, implementations *SHOULD* use environment variable references or external secret manager URIs rather than embedding values directly. Implementations *SHOULD* warn when string values match patterns commonly associated with credentials (e.g., values matching the format of known API key prefixes). Organizations *SHOULD* subject ADL documents to the same secret-scanning controls applied to source code repositories before storage or distribution.

18.3. Template Injection

The system_prompt member (Section 7.2) and prompts[*].template members (Section 8.3) support a template substitution syntax using {{variable_name}} placeholders. If variable values are derived from untrusted user input and substituted without sanitization, an attacker may be able to alter agent behavior by injecting malicious instructions into the rendered prompt -- including instructions that override the intended agent behavior or cause the agent to exfiltrate information.

Implementations *MUST* sanitize template variable values before substitution. At minimum, implementations *SHOULD* escape or reject values that contain the template delimiter sequence {{ or }}, and *SHOULD* apply length limits to variable values. Applications that allow end users to supply template variable values *SHOULD* treat such values as untrusted and apply content validation appropriate to the deployment context. Runtimes operating on agents with data_classification.sensitivity of confidential or restricted *SHOULD* log rendered prompts (after variable substitution) to enable post-incident review, subject to applicable privacy constraints.

18.4. Information Disclosure

ADL documents may reveal infrastructure details that are useful to attackers. The name, description, and tool description fields may disclose the existence of internal services or system architecture. The permissions.network.allowed_hosts list may reveal internal hostname patterns, private IP ranges, or internal service naming conventions. The permissions.filesystem.allowed_paths list may reveal sensitive directory structures. The provider.url, metadata.documentation, and metadata.repository fields may reference internal systems not intended for public visibility.

ADL documents intended for public distribution *SHOULD* be reviewed to remove or generalize infrastructure-specific information. Host patterns *SHOULD* use registered domain names rather than IP

addresses or internal hostnames. Path patterns *SHOULD* avoid exposing sensitive directory names. Documents with data_classification.sensitivity of confidential or restricted *SHOULD* only be distributed to parties with appropriate access authorization and *SHOULD NOT* be published to public registries without thorough review.

18.5. Resource Exhaustion

Implementations that parse and validate ADL documents are susceptible to resource exhaustion from adversarially crafted inputs. Specific attack vectors include: deeply nested JSON Schema in parameters and returns members (including circular $ref chains or exponentially expanding allOf/anyOf combinators); documents with very large numbers of tools, resources, or prompts; and documents with excessively long string values in system_prompt, description fields, or pattern arrays.

Implementations *SHOULD* enforce and document limits on: total document size (recommended maximum: 1 MB); JSON nesting depth (recommended maximum: 32 levels); number of entries in tools, resources, and prompts arrays (recommended maximum: 1000 each); string length for system_prompt and description fields (recommended maximum: 1 MB per field); and number of entries in any permission pattern array (recommended maximum: 500 patterns per domain). Implementations *SHOULD* terminate processing with an appropriate error code when any of these limits is exceeded rather than continuing to consume resources.

18.6. Pattern Matching Abuse

The permission pattern syntax (Section 4.4) governs access grants across network, filesystem, environment variable, and execution domains. Overly permissive patterns undermine the deny-by-default permission model; patterns that are expensive to evaluate can enable denial-of-service.

A bare * as the sole value of an entry in allowed_hosts grants access to all hostnames and effectively disables network permission enforcement. Implementations *MUST* warn when a bare * wildcard is used in any security-sensitive permission pattern, including allowed_hosts and allowed_variables. Implementations *SHOULD* require explicit user acknowledgment -- or refuse to deploy -- agents that use bare * patterns in these domains. Pattern evaluation *SHOULD* be bounded in time and space: implementations that use backtracking pattern matchers *SHOULD* reject or normalize patterns that would require exponential backtracking (e.g., consecutive wildcards such as ***). The ** multi-segment wildcard *MUST NOT* appear in host, environment, or command patterns, and implementations *MUST* reject documents in which it does.

18.7. URI Reference Attacks (SSRF)

Multiple ADL fields accept URI values: $schema, id, provider.url, metadata.documentation, metadata.repository, resource.uri, tool.annotations.openapi_ref, lifecycle.successor, security.attestation.issuer, and others defined by profiles. If an implementation automatically dereferences these URIs when processing a document from an untrusted source, an attacker may cause the implementation to issue requests to arbitrary endpoints, including internal services not reachable from the public internet -- a class of vulnerability known as Server-Side Request Forgery (SSRF).

Implementations *MUST NOT* automatically dereference URI values from ADL documents received from untrusted sources without explicit operator or user consent. Implementations that fetch external schema documents (e.g., via $schema) for validation purposes *SHOULD* use an allowlist of trusted schema hosts and *MUST NOT* follow redirects that leave the trusted set. When fetching openapi_ref documents for tool description or validation, implementations *SHOULD* verify that the target URI matches a pre-approved allowlist. Implementations *SHOULD* validate that URI values in ADL documents conform to [RFC3986] and *SHOULD* reject URIs with schemes other than https, http, or urn unless the deployment context explicitly allows them.

18.8. Canonicalization Attacks

ADL supports document integrity verification via cryptographic signatures using JCS canonicalization [RFC8785]. The security of this mechanism depends on all conforming implementations producing identical canonical byte sequences for the same logical document. Subtle differences in JCS implementations -- such as incorrect handling of Unicode escape sequences, floating-point number serialization, or object member ordering -- could cause a legitimate signature to fail verification, or, more critically, allow an

attacker to construct a document where different implementations produce different canonical forms, potentially enabling a signature verification bypass.

Implementations *MUST* use a conformant JCS [RFC8785] implementation for both signing and verification. Implementations *SHOULD* validate their JCS implementation against the test vectors provided in RFC 8785 before use in a production environment. Implementations *MUST NOT* verify signatures against non-canonical serializations such as pretty-printed JSON or YAML. Implementations that process ADL documents containing IEEE 754 floating-point values in signed content *SHOULD* be aware that platform-specific floating-point representation differences may affect canonicalization and *SHOULD* avoid floating-point values in fields that will be signed when possible.

18.9. Privacy Considerations

ADL documents may contain personal information subject to applicable privacy regulations. The provider.contact field (Section 6.2) contains a contact email address. The metadata.authors array (Section 12.1) may contain author names, email addresses, and URLs. The system_prompt member may contain information about intended user roles, user populations, or organizational context. When ADL documents are published to public registries or shared broadly, this information becomes publicly accessible.

Publishers *SHOULD* review ADL documents for personally identifiable information (PII) before public distribution and *SHOULD* use organizational or role-based contact addresses rather than personal email addresses. Implementations that log ADL document contents for debugging or auditing *SHOULD* redact or omit provider.contact, metadata.authors, and system_prompt fields from logs unless there is a documented operational requirement to retain them. Users *SHOULD* be informed when their ADL documents are transmitted to third-party services for validation, indexing, or registry queries.

18.10. Privilege Escalation

An ADL document that has been modified -- whether by a malicious actor during transmission or by a compromised storage or distribution system -- could grant an agent permissions or capabilities beyond those that were reviewed and approved for deployment. This risk is the primary motivator for the integrity mechanisms described in Section 10.3.

Implementations *SHOULD* verify document integrity (Section 10.3) before enforcing the permissions declared in a document, particularly when documents are retrieved from network locations, shared storage systems, or public registries. Runtimes that cannot verify document integrity *SHOULD* apply compensating controls -- such as mandatory human review -- before deploying agents that declare elevated permissions or sensitive data access. When a document's data_classification.sensitivity is confidential or restricted, runtimes *SHOULD* require a verified signature or a verified supply chain (e.g., document retrieved from a trusted registry over an authenticated and integrity-protected channel) before provisioning. Organizations *SHOULD* maintain an inventory of approved ADL documents along with their expected signatures or cryptographic digests, and *SHOULD* treat any discrepancy between the recorded and observed document as a potential security incident.

18.11. Cross-Origin and Supply Chain Concerns

ADL documents may be fetched from remote sources: registries, source control systems, artifact stores, or agent marketplaces. A document tampered with in transit or at the origin could cause a runtime to provision a malicious agent without the operator's knowledge.

ADL documents *SHOULD* be fetched over authenticated, integrity-protected channels (HTTPS with full certificate validation). Implementations *SHOULD* verify document signatures (Section 10.3) when documents are retrieved from remote or third-party sources. Implementations *SHOULD* validate that the signing identity declared in cryptographic_identity matches an expected, trusted identity for the document's declared provider.

Supply chain integrity requires attention at every reference boundary: the ADL document itself, referenced OpenAPI specifications (openapi_ref), and external JSON Schemas ($schema). Implementations that automatically resolve external references during provisioning *SHOULD* pin or verify all such references. When accepting ADL documents from third-party sources, implementations *SHOULD* apply an allowlist of trusted providers (based on provider.name or id URI authority), verify attestation signatures from trusted issuers, and treat documents from unverified sources with the same caution applied to untrusted executable code.

18.12. Permission Model and Defense in Depth

The deny-by-default permission model (Section 9.1) is a foundational security property of ADL: an agent can only access resources and capabilities that its ADL document explicitly permits. However, the effectiveness of this model depends entirely on the runtime correctly enforcing declared permissions. No permission model is a substitute for defense in depth.

Runtimes *MUST* enforce declared permissions and *MUST NOT* allow agents to exceed those permissions under any circumstances, including error conditions or fallback behaviors. Runtimes that cannot enforce a specific permission domain (e.g., because the underlying platform lacks the required isolation primitives) *MUST* warn users before execution and *SHOULD* refuse to execute the agent unless the user explicitly acknowledges the limitation.

Beyond permission enforcement, runtimes *SHOULD* monitor agent behavior during execution: logging tool invocations, recording network destinations contacted, and alerting on anomalous activity such as repeated attempts to access resources outside declared permissions. The ADL document represents intended access boundaries at definition time; runtime monitoring ensures actual behavior remains within those boundaries in production.

Runtimes *SHOULD* validate tool inputs and outputs against the declared JSON Schema (Section 8.1) before passing them to or from the agent. Malformed responses from external tool implementations could inject unexpected data into agent reasoning; runtime-level schema validation provides a defense against malfunctioning or malicious tool backends. Tools annotated with requires_confirmation: true *MUST* receive explicit user confirmation before invocation; runtimes *MUST NOT* invoke such tools autonomously regardless of other configuration.

Lifecycle status *MUST* be enforced as a security boundary. Runtimes *MUST NOT* provision or execute agents with lifecycle.status of retired. Retired agents may have revoked credentials, unpatched vulnerabilities, or stale permission configurations. Agents with lifecycle.status of deprecated *SHOULD* trigger warnings to operators, who *SHOULD* migrate to the agent identified by lifecycle.successor before the sunset_date is reached.

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>.

[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>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/rfc/rfc6901>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/rfc/rfc8141>.

[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>.

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

19.2. Informative References

[A2A] A2A Protocol Working Group, "Agent-to-Agent Protocol Specification", 2025, <https://a2a-protocol.org/specification>.

[AI-PROTOCOLS]
Rosenberg, J., "Framework, Use Cases and Requirements for AI Agent Protocols", Work in Progress, Internet-Draft, draft-rosenberg-ai-protocols-00, 2025, <https://datatracker.ietf.org/doc/html/draft-rosenberg-ai-protocols-00>.

[ISO-22989]
ISO/IEC JTC 1/SC 42, "Information technology -- Artificial intelligence -- Artificial intelligence concepts and terminology", ISO/IEC 22989:2022, 2022, <https://www.iso.org/standard/74296.html>.

[JSON-SCHEMA]
Wright, A., "JSON Schema: A Media Type for Describing JSON Documents", 2020, <https://json-schema.org/draft/2020-12/json-schema-core>.

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

[OPENAPI] OpenAPI Initiative, "OpenAPI Specification", 2024, <https://spec.openapis.org/oas/v3.1.0>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[W3C.DID] Sporny, M., "Decentralized Identifiers (DIDs) v1.0", 2022, <https://www.w3.org/TR/did-core/>.

[W3C.VC] Sporny, M., "Verifiable Credentials Data Model v1.1", 2022, <https://www.w3.org/TR/vc-data-model/>.

Appendix A. JSON Schema

The normative JSON Schema for ADL is available at https://adl-spec.org/0.2/schema.json (JSON Schema Draft 2020-12). A minimal required-fields schema is provided in schema.json (./schema.json) in this directory.

Appendix B. Examples

See the examples/ (./examples/) directory:

* *Minimal:* minimal.yaml (./examples/minimal.yaml)

* *Agent with tools:* with-tools.yaml (./examples/with-tools.yaml)

* *Production agent:* production.yaml (./examples/production.yaml)

Appendix C. Profiles

ADL profiles are maintained in the profiles/ (../../profiles/) directory. Each profile is versioned independently and declares compatibility with ADL versions.

C.1. Available Profiles

Profile	Identifier	Status
Governance (../../profiles/governance/overview)	urn:adl:profile:governance:1.0	Draft
Portfolio (../../profiles/portfolio/overview)	urn:adl:profile:portfolio:1.0	Draft
Healthcare (../../profiles/healthcare/overview)	urn:adl:profile:healthcare:1.0	Draft
Financial (../../profiles/financial/overview)	urn:adl:profile:financial:1.0	Draft

Table 22

See the profiles/ (../../profiles/) directory for the full profile index and contribution guidelines.

Appendix D. ABNF Grammar

This appendix defines formal ABNF grammar productions (RFC 5234 / RFC 7405) for syntactic constructs specified in this document. All productions use ASCII character references consistent with RFC 5234, Appendix B. The core ABNF rules ALPHA (letters), DIGIT (decimal digits), and VCHAR (visible ASCII characters) are defined in RFC 5234 Section 6 (B.1).

<CODE BEGINS> ‘‘‘abnf ; Semantic Versioning format (Sections 5.1, 5.5) semver = 1_DIGIT "." 1_DIGIT "." 1*DIGIT

```
; Tool name (Section 8.1) ; All alpha characters MUST be lowercase;
satisfies ^[a-z][a-z0-9_]*$ tool-name = lc-alpha *( lc-alpha / DIGIT
/ "_" ) lc-alpha = %x61-7A ; a-z (lowercase letters only)

; Vendor extension namespace key (Section 4.3) ; Reverse-domain
notation, minimum two segments vendor-key = domain-segment 1*("."
domain-segment) domain-segment = lc-alpha *(lc-alpha / DIGIT / "-")

; Template variable (Sections 7.2, 8.3) ; Used in system_prompt
templates and prompt templates template-var = 2%x7B var-name 2%x7D
var-name = ALPHA *( ALPHA / DIGIT / "_" ) ; First character MUST be a
letter (upper or lowercase)

; Tag (Section 12.5) ; Lowercase alphanumeric characters and hyphens
tag = 1*( lc-alpha / DIGIT / "-" )

; Pattern syntax (Section 4.4) ; An ADL pattern consists of literal
characters and optional wildcard tokens pattern = 1_pattern-element
pattern-element = multi-wildcard / single-wildcard / literal-chars
multi-wildcard = "**" ; Valid only in filesystem path patterns
(Section 9.3) ; MUST NOT appear in host, env-variable, or command
patterns single-wildcard = "_" ; Matches within one segment; does not
cross "." in host ; patterns or "/" in filesystem path patterns
literal-chars = 1_literal-char literal-char = %x21-29 / %x2B-7E ;
Printable ASCII except "_" (%x2A) ; "/" (%x2F) carries segment-
boundary meaning in path patterns ; "." (%x2E) carries segment-
boundary meaning in host patterns ''' <CODE ENDS>
```

D.1. Cross-Reference Summary

Production	Normative Section	Usage
semver	5.1, 5.5	adl_spec and version values
tool-name	8.1	Tool name values
vendor-key	4.3	Vendor extension namespace keys
template-var	7.2, 8.3	{{variable}} references in templates
tag	12.5	metadata.tags array items
pattern	4.4, 9.2-9.5	Permission domain pattern strings

Table 23

Acknowledgments

TBD

Author’s Address

Terrill Nederveld
Ironstead Group, LLC.
Email: terry+adl@ironsteadgroup.com