Independent Submission
Internet-Draft
Intended status: Informational
Expires: 19 October 2026

C. Hood
Nomotic, Inc.

17 April 2026

Agentic Grammar and Interface Specification (AGIS)
draft-hood-independent-agis-01

Abstract

This document defines the Agentic Grammar and Interface Specification (AGIS), a grammar-constrained interface definition language for Agentive APIs designed for consumption by large language model (LLM) based agents. AGIS establishes structural and semantic rules governing how API methods and endpoints must be expressed, without mandating a fixed vocabulary of method names. An implementation conforming to AGIS uses intent-expressing imperative verbs as method identifiers, enabling agents to infer operational meaning from method names without requiring training on a predefined catalog.

AGIS is the native interface definition layer for the Agent Transfer Protocol (AGTP) [AGTP], in the same way that HTML functions as the native content language for the HTTP transport protocol. AGIS does not replace JSON Schema [JSON-SCHEMA] for data contracts; it governs the structural grammar of method and endpoint design that wraps those contracts. AGIS-conformant methods are accepted at the AGTP transport layer via the Method-Grammar header without requiring prior IANA registration, enabling organizations to define domain-specific Agentive API vocabularies while preserving interoperability through shared grammatical constraints.

Empirical validation of the core AGIS design principle is provided in [HOOD2026], which demonstrates a 10-29 percentage point accuracy advantage for intent-expressing method names over generic HTTP verbs across three frontier LLM families in 7,200 controlled trials.

This version also introduces: normative YAML and JSON serialization formats; a Data Manifest block enabling services to declare available data without pre-built endpoints; a negotiable signal enabling AGTP dynamic endpoint instantiation; semantic declaration enhancements including is_idempotent, impact_tier, parameter_hints, and state_transition fields; vocabulary namespace disambiguation; and an HTTP transitional binding for incremental adoption.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Representational State Transfer (REST) architectural style [RFC7231] was designed for a specific class of consumer: the human developer. A developer reading "POST /reservations" brings domain knowledge, understands HTTP verb semantics, consults documentation, and infers intent from context. Large language model (LLM) based agents do not share this prior knowledge. They process method names and paths as natural language tokens, matching them against user intent in real time without the contextual scaffolding that human developers rely upon.

The consequence of this mismatch is empirically measurable. Agents consuming REST/CRUD interfaces exhibit substantially lower endpoint selection accuracy than agents consuming semantically equivalent

interfaces expressed with intent-aligned method verbs. The mechanism is the method name itself: generic HTTP verbs (GET, POST, PUT, DELETE) encode no intent-specific signal, requiring agents to infer purpose entirely from path structure and documentation text. Intent-expressing verbs (BOOK, FIND, SCHEDULE, RECONCILE) encode operational meaning directly, enabling agents to match user goals to endpoint functions with higher precision and greater resilience to documentation noise [HOOD2026].

1.1. The Natural Language Alignment Problem

Modern agentic systems operate on natural language instructions. A user says "schedule a meeting for Thursday" or "book a table at the restaurant." The agent must translate that natural language intent into a structured API call. The quality of that translation depends substantially on how much semantic distance exists between the user's words and the available method identifiers.

When the available method is POST /calendar/events, the agent must traverse a semantic chain: POST implies creation, /calendar/events implies calendar resource, and the combination implies event creation -- which may or may not match the user's intent to schedule a meeting. Each step in the chain is an inference opportunity for error.

When the available method is SCHEDULE /event, the semantic distance collapses. "Schedule a meeting" maps directly to SCHEDULE. The verb carries the intent. The noun confirms the resource. A single, low-ambiguity inference replaces a multi-step chain.

AGIS is designed to systematically minimize semantic distance between natural language user intent and API method identifiers. It does not prescribe which verbs to use. It prescribes the grammatical class that verbs must belong to -- the action-intent class -- which is precisely the class of verbs that appear naturally in user instructions. "Book", "find", "schedule", "cancel", "reconcile", "dispatch" are words users say. AGIS requires that API methods be words of this type.

Empirical research confirms that agents navigating unfamiliar AGIS-conformant service catalogs -- without knowing what methods exist in advance -- still performed better with semantic verbs than with CRUD. The agent did not need a dictionary. It inferred intent from the verb itself, the same way it infers meaning from any natural language word it encounters [HOOD2026]. This validates the grammar-over-vocabulary architecture: agents can reason about any AGIS-conformant verb through natural language inference, making a predefined method catalog unnecessary for agent comprehension.

1.2. Design Philosophy

AGIS encapsulates what every API exchange has always required: a method (what to do), authorization (who can do it), and a data contract (what goes in and comes out). The method is no longer GET or POST but BOOK or FIND. The authorization is negotiated contextually via AGTP [AGTP] rather than pre-issued as a static key. The data contract is JSON Schema, unchanged. AGIS is not a new idea. It is the right implementation of an old idea for a new class of consumer.

The Foundation for Intelligent Physical Agents Agent Communication Language (FIPA-ACL) is the direct historical precedent. FIPA-ACL defined approximately 25 fixed performatives -- REQUEST, INFORM, PROPOSE, ACCEPT-PROPOSAL, REJECT-PROPOSAL, QUERY-IF, CONFIRM -- each carried in a structured envelope with fields for sender, receiver, content, ontology, and conversation-id. The performative was the verb giving the message its intent. AGIS applies the same principle but replaces the fixed list of performatives with a grammar that accepts any action-intent verb. FIPA-ACL was built for symbolic AI agents in JADE frameworks. AGIS is its spiritual successor built for LLM agents that reason in natural language.

AGIS operates on a principle borrowed from formal grammar: define the rules of the language without prescribing the vocabulary. A grammar specification for English does not list every valid English word. It defines the rules -- syntactic structure, part-of-speech requirements, conjugation patterns -- under which any word is a valid member of the language.

In the same way, AGIS defines the rules under which any verb is a valid AGIS method. This approach directly addresses the principal objection to fixed method vocabularies: that a library of defined methods is either too large to manage or too small to accommodate domain diversity. AGIS eliminates the library entirely as a compliance mechanism. The grammar is the standard. Vocabulary is implementation-defined.

Consider the restaurant analogy. An organization’s published AGIS document is the menu: pre-declared endpoints the agent can invoke directly. The data_manifest block is the kitchen inventory: data classes the service holds even if no endpoint for them has been built yet. When an agent arrives and cannot find what it needs on the menu, it can ask -- through the AGTP negotiation protocol -- "do you have location data?" The service checks its kitchen (evaluates the data_manifest and authorization policy) and responds with a dynamically instantiated endpoint or a structured refusal. The agent does not need to know the full kitchen inventory in advance. It

needs to know the service exists, that it holds relevant data, and that off-menu requests are accepted. This is what negotiable: true and the data_manifest block declare.

The specification mandates:

* That methods be expressed as imperative base-form verbs

* That methods belong to the action-intent semantic class

* That methods be accompanied by machine-readable semantic declarations

* That implementations declare their vocabulary explicitly

* That all data contracts be expressed in JSON Schema [JSON-SCHEMA]

The specification does not mandate which verbs to use. Two organizations implementing AGIS-conformant APIs may use entirely different verb vocabularies. BOOK /reservation and RESERVE /booking are both valid AGIS method-path combinations expressing the same user intent category. An LLM agent encountering either service discovers and interprets it correctly through natural language inference -- the same inference capability demonstrated in two-stage discovery conditions in [HOOD2026].

Compliance validation follows the grammar checker model. A grammar checker does not validate that a writer chose the "right" words. It validates that the words used are grammatically correct. An AGIS validator performs the equivalent checks: that method identifiers are imperative base-form verbs, that they belong to the action-intent class, that semantic declarations are present and internally consistent, and that the document structure is well-formed. Five structural checks. Zero vocabulary judgments.

1.3. Relationship to AGTP

AGIS is the interface definition layer for the Agent Transfer Protocol [AGTP]. The relationship between AGIS and AGTP mirrors the relationship between HTML and HTTP:

Analogy	AGTP/AGIS Stack
HTTP	AGTP -- transport protocol for agent-to-service communication
HTML	AGIS -- interface definition language for Agentive APIs
Web browser	Agent runtime -- consumer of AGIS documents
Web page	AGIS service definition -- the artifact served at an AGTP address
W3C HTML Specification	This document -- the grammar specification

Table 1: AGIS/AGTP Structural Analogy

An AGTP address (agtp://service.example.com) serves an AGIS document as its root response. Agent runtimes retrieve, validate, and parse the AGIS document to build an internal service map before issuing method calls. The AGIS document is both the service contract and the discovery artifact.

AGTP version 03 [AGTP] introduces the Method-Grammar: AGIS/1.0 header, which instructs the AGTP transport layer to validate method identifiers against AGIS grammar rules rather than checking the IANA registry exclusively. This enables Tier 4 custom methods: organization-defined, AGIS-conformant verbs that are accepted at the transport layer without prior IANA registration. The Tier 1 and Tier 2 registered methods defined in [AGTP] and [AGTP-METHODS] are AGIS-conformant reference vocabulary and remain available for maximum cross-system interoperability.

1.4. AGIS in the 2026 Agent Protocol Stack

AGIS occupies a distinct and complementary role in the 2026 agent protocol ecosystem. Each layer solves a different problem:

Protocol	Role	Relationship to AGIS
AGTP	Transport, identity, governance, negotiation	AGIS is AGTP's native interface definition layer
MCP	Tool and context access for LLM runtimes	AGIS services auto-generate MCP tool entries (Appendix C)
A2A	Agent-to-agent task coordination	AGIS well-known endpoints enable A2A service discovery
AGIS	Service interface definition language	Grammar layer defining what services declare and how agents read it

Table 2: AGIS Role in the 2026 Agent Protocol Stack

AGIS does not compete with MCP, A2A, or AGTP. It is the missing service- contract layer that makes the ecosystem cohere. An organization publishes an AGIS document; it is immediately readable by AGTP-native agents, auto-convertible to MCP tool entries for Claude and Cursor, discoverable by A2A orchestrators via the well-known endpoint, and validatable by any AGIS-conformant linter. The grammar is the shared contract that all four protocol layers can consume.

1.5. Relationship to Standard Method Vocabulary

The AGTP Standard Extended Method Vocabulary [AGTP-METHODS] defines a reference set of registered AGIS-conformant methods for common agent operations. That document is the dictionary. This document is the grammar.

Organizations choosing methods from [AGTP-METHODS] gain interoperability across AGTP deployments. Organizations defining their own AGIS-conformant vocabularies gain domain specificity. Both approaches produce valid Agentive APIs. The grammar constraint is the shared foundation that makes both approaches machine-interpretable by agents.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174].

Agentive API: An API designed for consumption by LLM-based agents, implementing the AGIS grammar specification and served over the AGTP transport protocol. Practitioners building Agentive APIs are said to build "AG-APIs."

AGIS Document: A file conforming to this specification that describes the methods, paths, semantic declarations, and JSON Schema contracts of an Agentive API service. AGIS documents use the .agis file extension and the application/agis media type.

AGIS Grammar Specification: This document. The normative rules governing what constitutes a valid AGIS method, path, semantic declaration, and document structure.

Action-Intent Verb: An imperative base-form verb that expresses an operation the caller intends to be performed on their behalf. The semantic class required for all AGIS method identifiers. Action-intent verbs are the class of verbs users naturally employ in natural language instructions: "book", "find", "schedule", "cancel", "reconcile."

Imperative Base Form: The uninflected command form of a verb. BOOK is imperative base form. BOOKING (present participle), BOOKS (third-person singular), and BOOKED (past tense) are not. The form used when issuing instructions: "book a table", "find a restaurant", "schedule a meeting."

Method Identifier: The verb token used as an AGIS method name. Analogous to an HTTP method but constrained to the action-intent semantic class rather than fixed to a predefined set.

Semantic Declaration: A machine-readable block within an AGIS endpoint definition that describes intent, actor, and outcome in plain language. Used by agent runtimes for discovery, validation, and inference.

Vocabulary Block: A required section of an AGIS document that declares all method identifiers used within the document. Used by validators to verify declaration consistency and by agents to pre-index service capabilities.

Conforming Implementation: An AGIS document or service that satisfies all REQUIRED and MUST-level rules defined in this specification.

Agent Runtime: A software component that retrieves, validates, and consumes AGIS documents to enable LLM-based agents to discover and invoke Agentive API services.

Tier 4 Custom Method: An organization-defined method that is AGIS-conformant but not registered in the IANA AGTP Method Registry. Accepted at the AGTP transport layer via the Method-Grammar: AGIS/1.0 header. Discoverable by agents through the AGIS vocabulary block at the service's AGTP address.

FIPA-ACL: Foundation for Intelligent Physical Agents Agent Communication Language. The historical predecessor to AGIS, defining ~25 fixed performatives (REQUEST, INFORM, PROPOSE, ACCEPT-PROPOSAL, etc.) in structured message envelopes. Implemented in JADE and similar frameworks for symbolic AI agent systems. AGIS is FIPA-ACL's successor for LLM agents, replacing fixed performatives with a grammar-constrained open vocabulary.

Data Manifest: An optional block within an AGIS document that declares the data classes a service holds without defining pre-built endpoints. Enables agents to assess whether a service has relevant data before initiating dynamic endpoint negotiation via AGTP [AGTP]. The data manifest is the ceiling of what a service will negotiate; it does not constitute a commitment to serve any specific endpoint format.

Negotiable Service: An AGIS-conformant service that supports dynamic endpoint instantiation via the AGTP negotiation protocol. A negotiable service declares negotiable: true in its vocabulary block and MUST expose a data manifest describing available data classes.

Proposed Endpoint: An endpoint definition within an AGIS document that was dynamically instantiated during an AGTP negotiation session rather than pre-declared by the service. Proposed endpoints carry a proposed: true flag and are session-scoped unless the service explicitly persists them.

Impact Tier: A categorical classification of endpoint consequence used alongside confidence_guidance to enable model-independent escalation reasoning. Values are: Informational (read-only, no state change), Reversible (state change that can be undone), Irreversible (permanent state change or financial commitment).

State Transition: A machine-readable description of the expected system state change produced by successful endpoint execution. Used by Level 2 conformant agent runtimes for post-execution verification. Format: field: FROM_VALUE -> TO_VALUE.

3. AGIS Architecture

3.1. The Grammar Analogy

AGIS is a grammar specification, not a method dictionary. This distinction is architecturally fundamental and distinguishes AGIS from prior interface definition approaches that enumerate a fixed set of permitted operations.

A grammar specification for natural language defines syntactic rules, part-of-speech categories, and semantic class constraints without listing every valid word in the language. English grammar requires that verbs in imperative sentences be expressed in base form without specifying which verbs are permissible. "Book a table" and "reserve a table" are both grammatically valid despite using different verbs, because both verbs satisfy the grammatical class requirements of the imperative form.

AGIS applies this principle to API interface design. The specification defines that method identifiers MUST be imperative base-form verbs belonging to the action-intent semantic class. It does not enumerate which verbs satisfy this requirement. Any verb satisfying the grammatical and semantic class rules is a valid AGIS method identifier. Implementations choose their own vocabulary within these constraints.

This architecture directly eliminates the "too many methods" objection to semantic method vocabularies. The objection is a vocabulary objection: people argue about which specific verbs belong in a fixed list, whether 50 verbs is too many, whether LOCATE and FIND are redundant. AGIS removes the list from the standard entirely. There is no list to argue about. There is only a grammar. Any verb satisfying the grammar is valid.

3.2. Document Structure

An AGIS document is a structured text file using the .agis file extension. It contains four principal components:

Document Header: Version declaration, service name, AGTP address, and optional metadata.

Endpoint Definitions: One or more method-path combinations, each

accompanied by a semantic declaration block and JSON Schema contracts.

Schema Definitions: JSON Schema objects defining input parameters, output structures, and error states for each endpoint.

Vocabulary Block: An explicit declaration of all method identifiers used in the document, with domain classification.

The AGIS document is both the service contract and the discovery artifact. When an agent runtime retrieves an AGIS document from an AGTP address, it obtains a complete specification of what the service can do, what each operation means, what data each operation requires, and what each operation returns. No separate documentation is required for agent consumption.

3.3. Conformance Levels

This specification defines four conformance levels. All conforming implementations MUST satisfy Level 1. Higher levels are OPTIONAL and represent progressive adoption paths.

Level	Description	Status
Level 1 -- Declarative	AGIS document describes service interface. No runtime behavior.	REQUIRED
Level 2 -- Interpretable	AGIS document drives runtime routing and validation without additional implementation code.	OPTIONAL
Level 3 -- Scriptable	AGIS includes constrained expression layer for conditions and transformations.	OPTIONAL
Level 4 -- Executable	AGIS is a fully executable interface language with complete runtime semantics.	FUTURE

Table 3: AGIS Conformance Levels

This specification defines Level 1 conformance requirements. Guidance for Level 2 implementations is provided as informative content. Levels 3 and 4 are noted as future directions outside the scope of this document.

The following table summarizes which semantic declaration fields are REQUIRED, RECOMMENDED, or OPTIONAL at each conformance level:

Field	Level 1	Level 2	Notes
intent	REQUIRED	REQUIRED	All levels
actor	REQUIRED	REQUIRED	All levels
outcome	REQUIRED	REQUIRED	All levels
capability	RECOMMENDED	REQUIRED	Level 2 requires for routing
confidence_guidance	RECOMMENDED	REQUIRED	Level 2 requires for governance
impact_tier	RECOMMENDED	REQUIRED	Level 2 requires for escalation
is_idempotent	RECOMMENDED	REQUIRED	Level 2 requires for retry logic
parameter_hints	OPTIONAL	RECOMMENDED	Improves parametric accuracy
state_transition	OPTIONAL	REQUIRED	Level 2 requires for verification
mcp_tool_name	OPTIONAL	OPTIONAL	MCP bridge only

Table 4: Semantic Declaration Field Requirements by Conformance Level

4. Method Rules

4.1. Syntactic Requirements

An AGIS method identifier MUST satisfy the following syntactic rules:

(a) The method identifier MUST be a single token containing no whitespace characters.

(b) The method identifier MUST be expressed in the imperative base form of the verb. Inflected forms including present participle (-ing), past tense (-ed), and third-person singular (-s) are NOT PERMITTED.

(c) The method identifier MUST contain only uppercase alphabetic characters (A-Z). Numerals, hyphens, underscores, and special characters are NOT PERMITTED.

(d) The method identifier MUST NOT be a compound verb phrase. Multi-word constructions are NOT PERMITTED.

(e) The method identifier SHOULD be expressed in uppercase to distinguish it from path components. Implementations MAY use mixed case; validators MUST treat method comparison as case-insensitive.

Design rationale for single-token requirement: Compound verb forms such as CANCEL_BOOKING or GENERATE_REPORT are explicitly prohibited. The correct AGIS pattern separates verb and noun: CANCEL /booking, GENERATE /report. This separation preserves the semantic density of the method identifier: CANCEL alone carries a complete action-intent signal; CANCEL_BOOKING dilutes it with path information already conveyed by the noun. Single-token identifiers also maximize the cosine similarity between the method verb and the user’s natural language instruction -- "cancel my booking" maps directly to CANCEL, not to CANCEL_BOOKING -- which is the mechanism empirically validated in [HOOD2026]. Implementations that require compound semantics SHOULD use the AGIS path and semantic declaration rather than encoding them in the method identifier.

The following examples illustrate syntactic compliance:

Method Identifier	Status	Reason
BOOK	VALID	Single token, imperative base form, uppercase alphabetic
FIND	VALID	Single token, imperative base form, uppercase alphabetic
SCHEDULE	VALID	Single token, imperative base form, uppercase alphabetic
RECONCILE	VALID	Single token, imperative base form, uppercase alphabetic
BOOKING	INVALID	Present participle -- not imperative base form
BOOK_TABLE	INVALID	Contains underscore -- compound form not permitted
book-reservation	INVALID	Contains hyphen -- compound form not permitted
FindRestaurant	INVALID	Compound form -- single token required
SEARCH2	INVALID	Contains numeral -- alphabetic only

Table 5: Method Identifier Syntactic Examples

4.2. Semantic Class Requirements

Beyond syntactic validity, an AGIS method identifier MUST belong to the action-intent semantic class. This class is defined as follows:

Definition: A verb belongs to the action-intent semantic class if it expresses an operation that the caller intends to have performed on their behalf, where the subject of the operation is an agent acting for a user or system goal.

The action-intent class requirement serves the core design goal of AGIS: enabling agents to infer operational purpose from method identifiers using natural language inference, without requiring a predefined vocabulary catalog. A verb in the action-intent class corresponds directly to the class of verbs users employ when expressing intent in natural language instructions -- "book a table", "find a restaurant", "cancel my reservation", "schedule a meeting."

A verb does NOT belong to the action-intent semantic class if it:

* Describes a state or condition rather than an action (e.g., AVAILABLE, ACTIVE, OPEN, VALID)

* Describes a noun recast as a verb without expressing intent (e.g., DATA used as a verb)

* Expresses existence or being rather than doing (e.g., EXISTS, IS, HAS)

* Is inherently ambiguous as to whether it expresses agent intent or system state

The following examples illustrate semantic class compliance:

Method Identifier	Semantic Class Status	Reason
BOOK	Action-intent	Expresses intent to create a booking
FIND	Action-intent	Expresses intent to locate a resource
CANCEL	Action-intent	Expresses intent to terminate a booking
AUTHORIZE	Action-intent	Expresses intent to grant permission
TRIAGE	Action-intent	Expresses intent to prioritize and route
DISPATCH	Action-intent	Expresses intent to send or route a resource
RESERVE	Action-intent	Expresses intent to hold a resource
LOCATE	Action-intent	Expresses intent to find a specific resource
AVAILABLE	NOT action-intent	Describes state, not action
ACTIVE	NOT action-intent	Describes condition, not operation
EXISTS	NOT action-intent	Describes existence, not intent
PROCESS	Borderline	May express intent or describe system state; a more specific verb is RECOMMENDED

Table 6: Method Identifier Semantic Class Examples

Note that RESERVE and LOCATE are valid AGIS methods even though BOOK and FIND serve equivalent purposes in the AGTP registered vocabulary. AGIS does not prohibit synonymous verbs across implementations. Two organizations may use BOOK and RESERVE for the same operation type;

both are AGIS-conformant. An agent encountering RESERVE infers its meaning through natural language inference -- the same mechanism demonstrated in two-stage discovery conditions in [HOOD2026].

4.3. Prohibited Methods

The following method identifiers are explicitly prohibited in AGIS implementations regardless of syntactic or semantic class validity, to prevent ambiguity with HTTP methods and established protocol conventions:

Prohibited: GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS, CONNECT, TRACE

Rationale: These identifiers carry HTTP-specific semantics that conflict with the AGIS design principle. Their use would create ambiguity for agents transitioning between HTTP/REST and AGTP/AGIS service contexts.

5. Path Rules

The path component of an AGIS endpoint identifies the entity upon which the method operates. The following rules apply:

(a) The path MUST begin with a forward slash character (/).

(b) The first path segment MUST be a noun or noun phrase identifying the primary entity type. Verb forms are NOT PERMITTED in path segments.

(c) Multi-word path segments MUST use hyphen-separated lowercase notation (e.g., /patient-record, /delivery-order).

(d) Parameterized path segments identifying specific resource instances MUST use curly-brace notation (e.g., /reservation/{id}, /order/{order_id}).

(e) The path MUST NOT duplicate the method verb. A method of BOOK with a path of /book-reservation violates this rule.

(f) Query string parameters MUST NOT appear in path definitions. Variable inputs are defined in the JSON Schema input block.

The following examples illustrate path compliance:

Path	Status	Reason
/reservation	VALID	Noun, lowercase, begins with /
/restaurant/{id}	VALID	Noun with parameterized identifier
/patient-record/{id}	VALID	Hyphenated noun phrase with identifier
/calendar/event/{id}	VALID	Hierarchical noun path
/bookReservation	INVALID	Verb in path
/get-restaurant	INVALID	Verb in path
/reservation?type=dinner	INVALID	Query string in path definition
/RESERVATION	INVALID	Uppercase -- path segments MUST be lowercase

Table 7: Path Syntax Examples

6. Semantic Declaration Requirements

Every endpoint in an AGIS document MUST include a semantic declaration block. The semantic declaration provides machine-readable descriptions that agent runtimes use for service discovery, intent matching, and confidence-based execution decisions.

6.1. Required Fields

intent (REQUIRED): A single sentence in plain natural language describing what this endpoint does on behalf of the caller. The intent statement MUST be written in terms of the agent’s goal, not the system’s internal operation. "Books a restaurant reservation on behalf of the requesting agent" is a valid intent statement. "Inserts a record into the reservations table" is NOT a valid intent statement for AGIS purposes.

actor (REQUIRED): Identifies who or what initiates the action. MUST be one of:

```
   agent   -- The calling LLM-based agent initiates on behalf of a user
   user    -- A human user initiates directly
   system  -- An automated system process initiates
```

outcome (REQUIRED): A single sentence describing the state that exists after successful execution of this endpoint. The outcome statement MUST describe a verifiable post-condition. "A confirmed reservation record exists in the system with status CONFIRMED" is a valid outcome statement.

6.2. Recommended Fields

capability (RECOMMENDED): Classifies the endpoint within a controlled capability taxonomy. Enables agent runtimes to filter and route by capability class without parsing intent statements. MUST be one of the following if present:

```
   discovery     -- Endpoint locates or returns matching resources
   transaction   -- Endpoint creates or commits an irreversible action
   modification  -- Endpoint updates an existing resource
   retrieval     -- Endpoint returns information about a known resource
   analysis      -- Endpoint processes data and returns derived results
   notification  -- Endpoint triggers a communication or alert
```

confidence_guidance (RECOMMENDED): A floating-point value between 0.0 and 1.0 specifying the minimum agent confidence threshold RECOMMENDED before autonomous execution of this endpoint. Endpoints with irreversible consequences (EXECUTE, CANCEL, TRANSFER) SHOULD declare a confidence_guidance of 0.85 or higher. Discovery and retrieval endpoints MAY declare lower thresholds. Agent runtimes SHOULD surface escalation options to users when agent confidence falls below the declared threshold. This field directly addresses the confidence calibration failure documented in [HOOD2026], in which description mismatch conditions produced +60 percentage point overconfidence errors.

impact_tier (RECOMMENDED): A categorical consequence classification providing a model-independent complement to confidence_guidance. Enables agent runtimes to apply appropriate reasoning loops regardless of the specific floating-point value. MUST be one of the following if present:

```
   informational  -- Read-only; no system state change occurs
   reversible     -- State change that can be undone (e.g., cancel a booking)
   irreversible   -- Permanent state change or financial commitment
```

Endpoints declaring irreversible SHOULD also declare confidence_guidance: 0.85 or higher. Agent runtimes encountering an irreversible endpoint SHOULD require explicit user confirmation regardless of confidence score.

is_idempotent (RECOMMENDED): A boolean indicating whether multiple identical invocations of this endpoint produce the same result as a single invocation. Agents use this field to determine whether automatic retry is safe on network failure. BOOK and TRANSFER are typically not idempotent. FIND and QUERY are typically idempotent. Defaults to false if absent.

parameter_hints (RECOMMENDED): A map from parameter names to natural language phrases that commonly refer to those parameters in user instructions. Enables agents to map fuzzy natural language input (e.g., "next Tuesday", "for two people") to specific schema fields without requiring exact vocabulary match. Example:

```
parameter_hints:
  party_size: ["number of guests", "for N people", "party of N",
               "table for N"]
  datetime:   ["next Tuesday", "tomorrow night", "Saturday at 7",
               "this weekend"]
```

state_transition (RECOMMENDED for Level 2): A machine-readable description of the expected system state change produced by successful execution. Enables Level 2 conformant agent runtimes to verify post-execution outcomes without parsing the natural language outcome statement. Format: field: FROM -> TO. Example:

```
state_transition:
  reservation.status: NONE -> CONFIRMED
  reservation.id: NULL -> [assigned]
```

Level 2 conformance REQUIRES this field to be present on all endpoints that produce state changes.

mcp_tool_name (OPTIONAL): An override for the tool name generated when this endpoint is auto-converted to an MCP tools/list entry (see Appendix C). When absent, the default auto-generation rule applies: METHOD_pathsegment in lowercase (e.g., BOOK /reservation -> book_reservation). Use this field when the default name conflicts with an existing MCP tool in the same service catalog, or when a more descriptive name is needed for MCP client discoverability.

```
mcp_tool_name: make_reservation
```

The mcp_tool_name MUST be lowercase, MAY contain underscores, and MUST be unique within the service's generated MCP catalog. When parameter_hints are declared, they SHOULD be appended to the MCP tool description to enrich MCP client elicitation behavior.

6.3. Semantic Consistency Rule

The intent field of a semantic declaration MUST be semantically consistent with the method identifier. A validator MUST flag as non-conformant any endpoint where the stated intent contradicts the operation implied by the method verb.

Examples of consistency violations:

* Method: BOOK | Intent: "Returns a list of available restaurants" -- VIOLATION. BOOK implies creation/commitment; the intent describes retrieval.

* Method: FIND | Intent: "Creates a new reservation in the system" -- VIOLATION. FIND implies discovery; the intent describes creation.

* Method: CANCEL | Intent: "Retrieves cancellation policy" -- VIOLATION. CANCEL implies termination; the intent describes retrieval.

7. JSON Schema Contract Requirements

AGIS does not define a data contract language. JSON Schema [JSON-SCHEMA] is used for all input parameter and output response definitions. Every AGIS endpoint MUST include the following JSON Schema blocks:

input (REQUIRED): A JSON Schema object defining the parameters accepted by this endpoint. MUST distinguish required parameters from optional parameters using the JSON Schema "required" array. Parameter names SHOULD use snake_case.

output (REQUIRED): A JSON Schema object defining the structure of a successful response from this endpoint.

errors (REQUIRED): An array of named error conditions that this endpoint may return. Each error condition MUST be named and SHOULD include a description. Generic error names (e.g., "error", "failure") are NOT RECOMMENDED. Specific descriptive error names (e.g., "reservation_unavailable", "invalid_datetime", "restaurant_not_found") are REQUIRED for conformance.

The following illustrates a conformant JSON Schema block for an endpoint:

```
input:
  required:
    - restaurant_id  (integer)
    - party_size     (integer, minimum: 1, maximum: 20)
    - datetime       (string, format: date-time, ISO 8601)
  optional:
    - preferences    (string)
    - contact_name   (string)

output:
  reservation_id     (string)
  status             (string, enum: [CONFIRMED, PENDING])
  confirmation_code  (string)
  datetime           (string, format: date-time)

errors:
  - reservation_unavailable
  - invalid_datetime
  - restaurant_not_found
  - party_size_exceeds_capacity
```

8. Document Structure

8.1. Top-Level Fields

An AGIS document MUST contain the following top-level fields:

Field	Status	Description
agis	REQUIRED	AGIS specification version. Current value: "1.0"
service	REQUIRED	Human-readable service name
agtp	REQUIRED	The AGTP address at which this service is available
endpoints	REQUIRED	Array of one or more endpoint definitions
vocabulary	REQUIRED	Vocabulary block (see Section 8.2)
description	RECOMMENDED	Human-readable service description
version	RECOMMENDED	Implementation version of this service
schemas	OPTIONAL	Shared JSON Schema definitions referenced by endpoints

Table 8: AGIS Document Top-Level Fields

8.2. Vocabulary Block

Every AGIS document MUST include a vocabulary block at the document level. The vocabulary block declares all method identifiers used in the document and provides domain classification. This block enables validators to verify declaration consistency and enables agent runtimes to pre-index service capabilities before parsing individual endpoints.

The vocabulary block MUST satisfy the following:

(a) Every method identifier used in an endpoint definition MUST be listed in the declared_verbs array.

(b) Every method identifier listed in declared_verbs MUST be used in at least one endpoint definition. Declared but unused verbs constitute a non-conformance.

(c) The domain field SHOULD identify the primary service domain using a plain-language descriptor (e.g., hospitality, healthcare, financial-services, logistics).

(d) The optional namespace field provides a scoped qualifier to disambiguate verb semantics when agents hold multiple service maps simultaneously. Agents SHOULD use namespace to distinguish between identically-named verbs from different services (e.g., logistics.DISPATCH vs workflow.DISPATCH). Format: domain.VERB at runtime resolution.

(e) The optional negotiable field signals that this service supports dynamic endpoint instantiation via the AGTP negotiation protocol [AGTP]. When negotiable: true, the service MUST also provide a data_manifest block (see Section 8.3). When absent, negotiable defaults to false.

```
vocabulary:
  declared_verbs: [BOOK, FIND, CANCEL, CHECK]
  domain: hospitality
  namespace: acme-reservations
  version: "1.2.0"
  negotiable: true
```

8.3. Data Manifest Block

When negotiable: true is declared in the vocabulary block, the AGIS document MUST include a data_manifest block. The data manifest declares the data classes the service holds, without defining pre-built endpoints. This enables agents to assess whether the service has relevant data before initiating dynamic endpoint negotiation via AGTP [AGTP].

The data manifest is informative for the agent and normative for the service: a service MUST NOT negotiate endpoints for data classes absent from its manifest. The manifest defines the ceiling of negotiation, not the floor.

```
data_manifest:
  available_data:
    - class: location
      description: "Geographic coordinates and address data for entities"
      formats: [json, geojson]
      sensitivity: low
    - class: reservation
      description: "Booking records and availability windows"
      formats: [json]
      sensitivity: medium
    - class: customer-profile
      description: "Customer identity and preference data"
      formats: [json]
      sensitivity: high
      requires_authorization: true
  pre_auth_discovery: true
```

The pre_auth_discovery: true field signals that the data manifest and base AGIS document are accessible without credentials. Agents MUST be able to retrieve these without prior authorization. Authorization is established during AGTP negotiation, not before discovery.

Data class sensitivity values follow the same enumeration as impact_tier: informational, reversible, irreversible. High-sensitivity data classes SHOULD declare requires_authorization: true.

9. Validation

9.1. Validation Passes

An AGIS validator MUST perform the following validation passes in sequence. Failure at any pass MUST produce a specific, actionable error message identifying the non-conformant element and the violated rule.

Pass	Name	Description
1	Structural	Verifies all required top-level fields are present and non-empty
2	Method Syntax	Verifies all method identifiers are single uppercase alphabetic tokens in imperative base form
3	Method Class	Verifies all method identifiers belong to the action-intent semantic class and are not prohibited HTTP methods
4	Path Syntax	Verifies all paths begin with /, contain no verbs, and use correct parameterization syntax
5	Semantic Completeness	Verifies all semantic declaration blocks contain intent, actor, and outcome fields
6	Semantic Consistency	Verifies intent declarations are semantically consistent with method identifiers
7	Vocabulary Integrity	Verifies declared_verbs matches the set of verbs used in endpoint definitions
8	Schema Completeness	Verifies all endpoints reference valid input, output, and errors JSON Schema blocks

Table 9: AGIS Validation Passes

Passes 1 through 5 and Pass 7 are fully mechanical and require no semantic judgment. Pass 6 requires semantic consistency evaluation. Validators SHOULD implement Pass 6 using a natural language inference model rather than keyword matching. Pass 8 requires JSON Schema validation of referenced schema objects.

Pass 6 implementation guidance: validators SHOULD compute the cosine similarity between the embedding of the method identifier and the embedding of the intent statement. A similarity score below 0.3 (on a standard sentence-transformer scale) indicates likely inconsistency and SHOULD trigger a warning. A score below 0.1 SHOULD trigger a

hard failure. This threshold is informative; implementations MAY adjust based on their chosen embedding model. The reference validator (agis-validator, PyPI) uses sentence-transformers/all-MiniLM-L6-v2 as the default embedding model.

9.2. Validator Implementation Guidance

AGIS validators SHOULD be implemented as open-source tools to accelerate ecosystem adoption. The reference validator architecture consists of three components:

(a) A structural validator (Passes 1, 2, 4, 5, 7, 8): implemented in any language against the normative JSON Schema for AGIS documents. A Python reference implementation is provided in the agis-validator package (agis-validator on PyPI).

(b) A semantic class classifier (Pass 3): a lightweight binary classifier determining whether a method identifier belongs to the action-intent semantic class. The reference implementation uses a fine-tuned sentence-transformer model. A rule-based fallback using the prohibited methods list (Section 4.3) and a curated stoplist of common non-action-intent tokens (AVAILABLE, ACTIVE, EXISTS, STATUS, DATA, INFO) is RECOMMENDED as a first-pass filter.

(c) A semantic consistency evaluator (Pass 6): embedding similarity between method identifier and intent statement, as described above. An LLM-powered consistency check (submitting method + intent to a small local model for binary pass/fail) is RECOMMENDED as a secondary pass for borderline cases.

A VS Code extension and CLI tool for .agis file validation are provided via the agis-tools and agis-cli packages (PyPI). These tools are RECOMMENDED for organizations authoring AGIS documents before publication.

A document that passes all eight validation passes is a conforming AGIS document. Partial conformance is not defined; an AGIS document either conforms or does not.

When an AGTP request carries the Method-Grammar: AGIS/1.0 header, the AGTP transport layer runs Pass 2 and Pass 3 against the method identifier in the request. Failure returns status 454 Grammar Violation per [AGTP]. Full document validation is performed by the service endpoint upon receiving a complete AGIS document at the AGTP root address.

9.3. Conformance Testing

Reference conformance test cases for each validation pass are maintained alongside the agis-validator package (PyPI). Implementers of AGIS validators are RECOMMENDED to verify their implementations against the reference test suite before deployment.

A conforming AGIS validator MUST:

(a) Accept all documents in the reference valid document set without producing errors.

(b) Reject all documents in the reference invalid document set, producing specific error messages identifying the violated rule in each case.

(c) Produce no false positives on the reference borderline document set, where borderline documents are syntactically valid but semantically questionable.

10. AGIS File Format

AGIS documents use the .agis file extension and the application/agis media type. This specification defines two normative serialization formats: YAML and JSON. Implementations MUST accept both. Validators MUST process both without preference.

YAML is the RECOMMENDED format for human-authored AGIS documents due to its readability. JSON is the RECOMMENDED format for machine-generated AGIS documents and for transmission over AGTP. Both serializations MUST produce semantically identical documents when parsed.

Canonical rules:

- All field names are lowercase snake_case
- Method identifiers are uppercase (BOOK, FIND)
- Strings use double quotes in JSON; bare or quoted in YAML
- Boolean values: true / false (lowercase)
- Floating-point values: standard decimal notation (0.85)
- Arrays: JSON array syntax in JSON; YAML sequence syntax in YAML

A formal JSON Schema for the AGIS document structure will be maintained alongside the agis-validator package. This JSON Schema is normative. AGIS validators MUST validate documents against this schema as part of Pass 1 (Structural) validation before proceeding to subsequent passes.

10.1. HTTP Transitional Binding

Organizations operating HTTP/REST services that wish to adopt AGIS semantics without full AGTP migration MAY use the HTTP transitional binding. This binding allows AGIS-conformant method identifiers to be carried in HTTP requests via a header:

```
X-AGIS-Method: BOOK
X-AGIS-Grammar: AGIS/1.0
X-AGIS-Namespace: acme-reservations
```

When these headers are present on an HTTP request, the service SHOULD interpret the method identifier according to AGIS grammar rules rather than HTTP method semantics. The HTTP method (e.g., POST) continues to govern transport behavior. The X-AGIS-Method governs semantic routing.

The optional X-AGIS-Namespace header disambiguates the method when the client holds multiple service maps simultaneously, corresponding to the namespace field in the vocabulary block. This is particularly relevant when multiple services use identically-named verbs (e.g., DISPATCH in both a logistics and a workflow service).

This transitional binding is informative. Full AGIS conformance requires AGTP [AGTP] as the transport layer. The HTTP binding is provided to lower the adoption barrier for incremental migration.

The following is a complete conformant AGIS document example in illustrative notation:

```
agis: "1.0"
service: "Acme Restaurant Reservations"
agtp: "agtp://reservations.acme.com"
description: "Reservation management for restaurant booking agents"

endpoints:

  BOOK /reservation {
    semantic {
      intent:   "Books a restaurant reservation on behalf of the
                 requesting agent"
      actor:    agent
      outcome:  "A confirmed reservation record exists with
                 status CONFIRMED"
      capability: transaction
      confidence_guidance: 0.85
    }
    input {
```

```
        required: [restaurant_id (int), party_size (int),
                   datetime (ISO8601)]
        optional: [preferences (string), contact_name (string)]
      }
      output { reservation_id, status, confirmation_code, datetime }
      errors { reservation_unavailable, invalid_datetime,
               restaurant_not_found }
    }

    FIND /restaurants {
      semantic {
        intent:   "Finds restaurants matching the agent criteria"
        actor:    agent
        outcome:  "A ranked list of matching restaurants is returned"
        capability: discovery
        confidence_guidance: 0.60
      }
      input {
        required: [location (string)]
        optional: [cuisine, party_size, datetime, price_range]
      }
      output { restaurants: [array of restaurant objects] }
      errors { no_results, invalid_location }
    }

  vocabulary {
    declared_verbs: [BOOK, FIND]
    domain: hospitality
    version: "1.0.0"
  }
```

An organization using domain-specific vocabulary would follow the same structure with different verbs. A healthcare service might use:

```
agis: "1.0"
service: "Acme Patient Services"
agtp: "agtp://patients.acme-health.com"

endpoints:

  TRIAGE /patient {
    semantic {
      intent:   "Assesses and prioritizes a patient for care"
      actor:    agent
      outcome:  "A triage record exists with assigned priority level"
      capability: transaction
      confidence_guidance: 0.92
    }
    ...
  }

  ADMIT /patient {
    semantic {
      intent:   "Admits a patient to a care facility"
      actor:    agent
      outcome:  "Patient record shows admitted status with assigned unit"
      capability: transaction
      confidence_guidance: 0.95
    }
    ...
  }

vocabulary {
  declared_verbs: [TRIAGE, ADMIT]
  domain: healthcare
  version: "1.0.0"
}
```

Neither TRIAGE nor ADMIT appears in the AGTP registered vocabulary, yet both are valid AGIS-conformant methods. An agent encountering this service via Method-Grammar: AGIS/1.0 at the AGTP transport layer can discover, validate, and invoke these methods through natural language inference against the semantic declarations, without a predefined catalog.

A financial services example demonstrates the data manifest and negotiation features together. The service has no pre-built endpoint for transaction history, but declares the data class as available for negotiation:

```
agis: "1.0"
service: "Acme Financial Services"
agtp: "agtp://api.acme-finance.com"

endpoints:

  TRANSFER /funds {
    semantic {
      intent:   "Transfers funds between accounts on behalf of the agent"
      actor:    agent
      outcome:  "Funds are debited from source and credited to destination"
      capability: transaction
      confidence_guidance: 0.95
      impact_tier: irreversible
      is_idempotent: false
      state_transition:
        source_account.balance: [current] -> [current - amount]
        destination_account.balance: [current] -> [current + amount]
    }
    input {
      required: [source_account_id, destination_account_id, amount,
                 currency]
      optional: [memo, scheduled_date]
    }
    output { transfer_id, status, timestamp, confirmation_code }
    errors { insufficient_funds, account_not_found, limit_exceeded,
             currency_mismatch, authorization_required }
  }

  RECONCILE /account/{id} {
    semantic {
      intent:   "Reconciles account records against transaction ledger"
      actor:    agent
      outcome:  "Discrepancies are identified and a reconciliation
                 report is returned"
      capability: analysis
      confidence_guidance: 0.75
      impact_tier: informational
      is_idempotent: true
    }
    input {
      required: [id]
      optional: [from_date, to_date, include_pending]
    }
    output { reconciliation_id, discrepancies: [array], status,
             period_covered }
    errors { account_not_found, period_too_large }
  }
```

```
vocabulary {
  declared_verbs: [TRANSFER, RECONCILE]
  domain: financial-services
  namespace: acme-finance
  version: "2.1.0"
  negotiable: true
}

data_manifest {
  available_data:
    - class: transaction-history
      description: "Full transaction ledger for authorized accounts"
      formats: [json, csv]
      sensitivity: high
      requires_authorization: true
    - class: exchange-rates
      description: "Real-time and historical currency exchange rates"
      formats: [json]
      sensitivity: informational
      requires_authorization: false
  pre_auth_discovery: true
}
```

An agent needing transaction history arrives, finds no pre-built endpoint, reads the data manifest, and sends a PROPOSE request for a RETRIEVE /account/{id}/transactions endpoint. The service evaluates the proposal against the transaction-history data class, establishes authorization for the requesting agent, and returns a 263 Endpoint Instantiated response with a session-scoped endpoint definition. The agent calls the endpoint without the service ever having pre-built it.

11. Agent Discovery Protocol

When an agent runtime issues a discovery request to an AGTP address, the service MUST return a valid AGIS document as the root response. The agent runtime MUST:

(a) Retrieve the AGIS document from the root AGTP address.

(b) Validate the document against the AGIS Grammar Specification. Non-conformant documents MUST be treated as discovery failures.

(c) Parse the vocabulary block to build an initial service capability index.

(d) Parse individual endpoint definitions to build a complete service map.

(e) Match incoming user intent against the service map using natural language inference against intent declarations and method identifiers.

Because AGIS method identifiers are constrained to the action-intent semantic class, agent runtimes SHOULD match incoming user natural language instructions against method identifiers directly before consulting intent declarations. The method identifier serves as a compressed intent signal; the intent declaration provides confirmation and disambiguation.

For example, a user instruction of "schedule a meeting for Thursday" would match as follows:

Step 1: Method matching -- agent scans vocabulary block for action-intent verbs semantically proximate to "schedule". SCHEDULE, BOOK, ARRANGE are candidate matches.

Step 2: Path matching -- agent evaluates path nouns against "meeting" intent. /event, /meeting, /calendar-event are candidate matches.

Step 3: Intent confirmation -- agent reads intent declarations for candidate endpoints to confirm alignment.

Step 4: Parameter construction -- agent maps "Thursday" to the required datetime parameter using the JSON Schema type constraint (ISO 8601 date).

Step 5: Confidence evaluation -- agent compares its selection confidence against the endpoint’s confidence_guidance. If below threshold, escalation is surfaced to the user.

This discovery process was empirically validated in the two-stage discovery conditions of [HOOD2026]. Agents navigating unfamiliar AGIS-conformant service catalogs demonstrated higher endpoint selection accuracy than agents navigating equivalent REST/CRUD catalogs, confirming that the AGIS grammar constraint -- without a fixed vocabulary -- provides sufficient semantic signal for effective agent discovery.

11.1. Well-Known Discovery Endpoint

AGIS-conformant services SHOULD expose a lightweight discovery summary at the well-known URI /.well-known/agis.json. This endpoint enables agents and agent registries to discover the service's capabilities without fetching the full AGIS document. The well-known endpoint parallels the agent-card.json pattern used in agent-to-agent protocols.

The well-known response MUST be a JSON object containing:

```
{
  "agis": "1.0",
  "service": "Acme Restaurant Reservations",
  "agtp": "agtp://reservations.acme.com",
  "agis_document": "agtp://reservations.acme.com",
  "methods": ["BOOK", "FIND", "CANCEL"],
  "domain": "hospitality",
  "namespace": "acme-reservations",
  "negotiable": true,
  "capability_summary": ["transaction", "discovery", "modification"],
  "data_classes": ["reservation", "restaurant", "availability"],
  "pre_auth_discovery": true,
  "version": "1.2.0",
  "interaction_protocols": ["request", "negotiate", "confirm"],
  "related_services": [
    "agtp://payments.acme.com",
    "agtp://loyalty.acme.com"
  ],
  "mcp_tools_list": "agtp://reservations.acme.com/mcp/tools"
}
```

The interaction_protocols array lists the communication patterns this service supports, borrowing the interaction protocol concept from FIPA-ACL [FIPA-IP]. Defined values are: request (standard method invocation), negotiate (dynamic endpoint instantiation via PROPOSE), confirm (multi-step confirmation flow), and query (read-only retrieval patterns). This enables A2A orchestrators to select services based on workflow compatibility.

The related_services array enables service mesh discovery: agents that find one AGIS service can follow links to discover related services in the same ecosystem without a central registry.

The mcp_tools_list field provides the URL of the service's MCP-compatible tools/list endpoint. When present, MCP clients can discover and invoke the service without reading the full AGIS document.

AGIS services that expose this endpoint become first-class participants in A2A agent workflows. A2A orchestrators discovering AGIS services via the well-known endpoint can treat them as invocable task participants, using the methods and capability_summary to route tasks appropriately.

This endpoint:

(a) MUST be accessible without credentials when pre_auth_discovery: true is declared in the full AGIS document.

(b) MUST NOT expose data_manifest sensitivity details or customer data.

(c) SHOULD be indexed by AGTP registries and A2A agent discovery mechanisms to enable cross-protocol service discovery.

(d) MUST reference the full AGIS document location via the agis_document field so agents can fetch the complete contract when needed.

Services implementing this endpoint become discoverable by any agent runtime that supports the well-known discovery pattern, including A2A agents, MCP clients, and AGTP-native agents, regardless of which protocol they use for subsequent interaction.

12. Empirical Foundation

The design requirements of this specification are grounded in empirical research conducted across 7,200 controlled trials spanning four LLM families (Claude Sonnet 4.6, Grok-3, GPT-4o, and Llama 3.2 3B) and 18 experimental conditions [HOOD2026]. Key findings supporting specific design decisions are summarized below.

Design Decision	Empirical Support
Method identifiers MUST belong to the action-intent semantic class	Intent-expressing verbs outperformed REST/CRUD methods by 10-29pp in mixed-paradigm conditions (z=3.77, p<0.001)
Grammar constraint rather than fixed vocabulary	Two-stage discovery conditions demonstrated agents successfully navigate unfamiliar semantic verb vocabularies through inference, without a predefined catalog
Semantic declarations MUST include intent field	Description-swap ablation showed intent-method consistency failures collapse accuracy 39-43pp; intent declarations mitigate this risk
confidence_guidance field RECOMMENDED for high-consequence endpoints	Description mismatch conditions produced +60pp calibration error; confidence guidance enables governance-aware execution
Effect requires frontier-scale reasoning	Llama 3.2 (3B parameters) showed zero accuracy advantage from semantic naming; capability threshold lies above 3B parameters
Vocabulary block enables pre-indexing	Discovery recall was higher when agents could filter by semantic class before full endpoint evaluation

Table 10: Empirical Support for AGIS Design Decisions

The research established that the tested vocabulary -- BOOK, FIND, QUERY, SUMMARIZE -- is an instance of the action-intent semantic class that AGIS standardizes, not a proposed canonical list. The performance advantage is a property of the semantic class, generalizing to any conformant vocabulary an implementation chooses. This is the empirical foundation for the grammar- over-vocabulary architecture: the mechanism is the semantic class, and any verb in that class produces the same accuracy advantage.

13. Security Considerations

AGIS-conformant services operate within the AGTP security model defined in [AGTP]. The following security considerations are specific to the AGIS layer.

Intent Statement Injection: Malicious actors may construct AGIS documents with intent statements designed to mislead agent runtimes into invoking unintended endpoints. Agent runtimes MUST validate AGIS documents against the Grammar Specification before consuming intent declarations. Non-conformant documents MUST be rejected.

Vocabulary Spoofing: An attacker serving a non-conformant AGIS document at a legitimate AGTP address may attempt to redirect agents to incorrect endpoints by declaring misleading vocabulary. Agent runtimes SHOULD verify vocabulary declarations against actual endpoint definitions as part of Pass 7 validation.

Confidence Threshold Manipulation: A service declaring artificially low confidence_guidance values may induce agents to execute high-consequence operations without appropriate escalation. Agent runtimes SHOULD apply minimum confidence thresholds independent of service-declared values for operations the runtime classifies as high-consequence.

Schema Injection: JSON Schema objects within AGIS documents are subject to the security considerations of [JSON-SCHEMA]. Agent runtimes MUST validate JSON Schema blocks before using them to construct API calls.

Grammar Validation at Transport Layer: The Method-Grammar: AGIS/1.0 header instructs AGTP infrastructure to perform Pass 2 and Pass 3 validation at the transport layer. Transport- layer validators MUST reject requests with method identifiers that fail AGIS grammar rules with status 454 Grammar Violation per [AGTP]. This prevents non-conformant method identifiers from reaching application code.

Intent Statement Prompt Injection: Beyond false intent declarations,

attackers may attempt to embed prompt injection payloads within intent and outcome statements, targeting agent runtimes that process these fields through LLM inference. Validators SHOULD enforce length limits on intent and outcome fields (RECOMMENDED maximum: 500 characters). Agent runtimes SHOULD treat intent statements as structured data fields, not as natural language prompts to be executed. Similarity bounds between method identifier and intent statement SHOULD be enforced; statements with cosine similarity below 0.3 against the method verb embedding SHOULD be flagged.

Agent runtimes that feed AGIS intent and outcome fields to LLM inference engines SHOULD apply content filtering or sandboxing before processing. Recommended mitigations: (1) validate that intent fields contain no instruction-pattern tokens ("ignore previous instructions", "you are", "system:") before LLM processing; (2) process intent fields in an isolated inference context with a constrained system prompt that prohibits instruction following; (3) compare intent field embeddings against a reference distribution of known-safe intent statements and flag outliers for human review before the service is published.

Data Manifest Security: Services exposing pre_auth_discovery: true MUST ensure their data manifest reveals no sensitive data in class descriptions. Manifests are accessible without credentials and MUST be treated as public-facing documents. The manifest describes data categories, not data contents. Specific record counts, schema details, and field names SHOULD be withheld from the unauthenticated manifest and disclosed only after AGTP authorization.

Internationalization Note: While this specification uses English-language imperative verbs as examples, agents handling non-English user intent MAY map through embedding similarity, enabling multilingual deployments without requiring non-English method identifiers. The action-intent semantic class is language-independent at the validator level; a verb in any language that satisfies the syntactic rules (single uppercase alphabetic token, imperative base form) and semantic class requirements is a valid AGIS method identifier.

AGIS validators are English-centric on syntax: the uppercase A-Z character restriction means method identifiers are expressed as romanized uppercase tokens even in non-English deployments. Agents handling Japanese, Arabic, Chinese, or other script user intent perform embedding-space mapping to the romanized AGIS method identifier during discovery. The parameter_hints field SHOULD include non-English natural language phrases where the primary user population is non- English-speaking, as these phrases directly improve parametric accuracy for multilingual agents.

14. IANA Considerations

This document requests the following registrations from IANA:

Media Type Registration: application/agis

Type name:	application
Subtype name:	agis
Required parameters:	none
Optional parameters:	version
Encoding considerations:	UTF-8
Security considerations:	See Section 12
Published specification:	This document

File Extension Registration: .agis

Extension:	agis
Media type:	application/agis
Description:	Agentic Grammar and Interface Specification document
Published specification:	This document

15. References

15.1. Normative References

[AGTP] Hood, C., "Agent Transfer Protocol (AGTP)", Work in Progress, Internet-Draft, draft-hood-independent-agtp-03, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-independent-agtp-03>.

[JSON-SCHEMA] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", Work in Progress, Internet-Draft, draft-bhutton-json-schema-01, 2020, <https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-01>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/rfc/rfc7231>.

[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>.

15.2. Informative References

[AGTP-METHODS] Hood, C., "AGTP Standard Extended Method Vocabulary", Work in Progress, Internet-Draft, draft-hood-agtp-standard-methods-01, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-standard-methods-01>.

[FIELDING] Fielding, R. T., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Dissertation University of California, Irvine, 2000.

[HOOD2026] Hood, C., "Semantic Method Naming and LLM Agent Accuracy: A Controlled Benchmark of REST/CRUD versus Agentive API Interface Design", Working Paper Available by request. March 2026., 2026.

Appendix A. Recommended Starter Vocabulary (Informative)

This appendix provides a non-normative list of recommended starter verbs for common agent operations. This list does not constitute a compliance requirement. Any AGIS-conformant verb satisfying the grammar rules of Section 4 is equally valid. Organizations are encouraged to use this vocabulary for maximum cross-system interoperability and to consult [AGTP-METHODS] for the full registered Tier 1 and Tier 2 vocabulary.

Verb	Category	Common Use
BOOK	Transaction	Reserve a resource, seat, or time slot
FIND	Discovery	Locate matching

		resources by criteria
SCHEDULE	Transaction	Plan a future action or appointment
CANCEL	Transaction	Terminate a reservation or commitment
QUERY	Retrieval	Retrieve information by parameters
SUMMARIZE	Analysis	Synthesize content into condensed form
AUTHORIZE	Transaction	Grant permission for an operation
TRANSFER	Transaction	Move a resource between parties
SUBMIT	Transaction	Deliver a document or form for processing
NOTIFY	Notification	Send information to a recipient
VERIFY	Retrieval	Confirm the validity of a claim or record
DISPATCH	Transaction	Send or route a resource to a destination
LOCATE	Discovery	Find the position or address of an entity
RESERVE	Transaction	Hold a resource without confirming
TRIAGE	Transaction	Assess and prioritize for action
RECONCILE	Analysis	Identify and resolve discrepancies
RECOMMEND	Analysis	Propose options based on criteria
ESCALATE	Orchestration	Defer to a higher

		authority
EXTRACT	Analysis	Pull specific data from a document or record
VALIDATE	Retrieval	Confirm that data or a claim meets requirements
GENERATE	Analysis	Produce a new document, report, or artifact
MONITOR	Orchestration	Observe a resource or process over time
APPROVE	Transaction	Grant formal acceptance of a request
REJECT	Transaction	Formally decline a request with reason

Table 11: Recommended Starter Vocabulary (Non-Normative)

Common synonyms: RESERVE (equivalent scope to BOOK), LOCATE (equivalent scope to FIND), RETRIEVE (equivalent scope to QUERY), REVOKE (equivalent scope to CANCEL in access-control contexts). Any synonym satisfying the action-intent semantic class is equally valid. Organizations SHOULD choose the verb whose natural language form most closely matches their users' instructions.

For domain-specific verbs not shown here, consult [AGTP-METHODS] for the full registered Tier 1 and Tier 2 vocabulary, which includes FETCH, SEARCH, SUBMIT, TRANSFER, PURCHASE, AUTHORIZE, SYNC, ROUTE, and thirty additional registered methods organized by category.

Appendix B. Comparison with Existing Approaches (Informative)

This appendix positions AGIS relative to adjacent specifications and approaches to clarify its role as the interface definition layer for Agentive APIs.

OpenAPI / Swagger: OpenAPI defines REST/CRUD endpoint contracts

using HTTP methods and resource paths. It is the de facto standard for human-developer-facing API documentation. AGIS differs in two respects: it governs the semantic class of method identifiers (not just structure), and it is designed for agent consumption rather than developer consumption. OpenAPI documents can be translated to AGIS documents; see the mapping guidance below.

LLM Function Calling (OpenAI, Anthropic): Function calling schemas define callable tools for LLM agents using name, description, and parameter schemas. AGIS extends this model by adding transport-level identity (via AGTP), grammar constraints on method identifiers, confidence guidance, impact tiers, and negotiation support. AGIS endpoint definitions are structurally similar to function calling schemas but operate at the protocol level rather than the application layer.

GraphQL: GraphQL allows clients to specify exactly the data shape they need. AGIS and AGTP address a complementary problem: what operation the agent wants to perform (method semantics), not what fields it wants returned (data projection). A service could expose both GraphQL for data shaping and AGIS/AGTP for agent interaction semantics.

Model Context Protocol (MCP): MCP defines tool invocation semantics for agents running within LLM contexts. MCP operates at the messaging layer over HTTP. AGTP provides the transport substrate for MCP (see [AGTP]); AGIS provides the interface definition language for services MCP tools invoke.

A2A / ACP: Agent-to-Agent and Agent Communication Protocols define how agents communicate with each other. AGIS defines how agents communicate with services. These are complementary: an A2A message may instruct an agent to invoke an AGIS/AGTP service.

FIPA-ACL / KQML: FIPA-ACL (Foundation for Intelligent Physical Agents Agent Communication Language) is the direct historical predecessor to AGIS. FIPA-ACL defined approximately 25 fixed performatives -- REQUEST, INFORM, PROPOSE, ACCEPT-PROPOSAL, REJECT-PROPOSAL, QUERY-IF, CONFIRM, FAILURE -- each carried in a structured envelope with sender, receiver, content, ontology, protocol, and conversation-id fields. The performative was the verb giving the message its intent, analogous to the AGIS method identifier.

FIPA-ACL also defined reusable interaction protocols (Contract Net, Query Protocol, Request Protocol) that serve as conversation templates -- the conceptual ancestor of the AGTP negotiation protocol.

AGIS differs from FIPA-ACL in three respects: (1) AGIS uses an open grammar rather than a fixed performative set, enabling domain-specific vocabularies without standardization overhead; (2) AGIS uses JSON Schema rather than formal content languages (SL, KIF, RDF), aligning with contemporary web tooling; (3) AGIS is designed for LLM agents that reason in natural language, whereas FIPA-ACL was designed for symbolic AI agents in frameworks like JADE. FIPA-ACL was ahead of its time. AGIS is its successor built for the agents that exist today.

OpenAPI to AGIS Mapping (Informative):

OpenAPI field	AGIS equivalent
operationId	Method identifier (rewrite to imperative base form)
summary	intent statement
x-outcome (custom)	outcome statement
parameters	input.required / input.optional
requestBody	input schema
responses.200	output schema
responses.4xx	errors array
tags[0]	domain (vocabulary block)

Organizations migrating from OpenAPI SHOULD rewrite operationId values to imperative base form (createReservation -> BOOK, searchRestaurants -> FIND) as the primary migration step. The resulting AGIS document will be AGIS-conformant if the rewritten identifiers satisfy the action-intent semantic class requirements of Section 4.

Appendix C. MCP Interoperability Bridge (Informative)

The Model Context Protocol (MCP) is the most widely deployed agent tool interface as of 2026, supported natively by Claude, Cursor, VS Code, and an expanding ecosystem of agent runtimes. MCP uses JSON-RPC 2.0 with a tools/list endpoint that returns a catalog of callable tools, each with a name, description, and input schema.

AGIS and MCP are complementary rather than competing. MCP is tool-centric and runtime-scoped; AGIS is service-centric and transport-governed. An AGIS document can automatically generate a valid MCP tools catalog, making any AGIS-conformant service immediately consumable by every MCP client without additional development.

C.1. AGIS to MCP Mapping

AGIS field	MCP equivalent
Method identifier (e.g., BOOK)	tool.name (e.g., "BOOK_reservation" or "book")
intent statement	tool.description
input.required + optional	tool.inputSchema (JSON Schema)
output	(MCP response body; no direct mapping)
errors	(MCP error codes; no direct mapping)
confidence_guidance	(informative; no MCP equivalent)
impact_tier	(informative; no MCP equivalent)

C.2. Auto-Generation Rules

An AGIS-to-MCP converter MUST apply the following rules:

(a) Tool name: if mcp_tool_name is declared in the semantic block, use it directly. Otherwise concatenate the method identifier and the first path segment, separated by underscore and lowercased. BOOK /reservation -> book_reservation BOOK /reservation with mcp_tool_name: make_reservation -> make_reservation

(b) Tool description: use the AGIS intent statement as the base description. If parameter_hints are declared, append them to the description in a structured format to enrich MCP client elicitation:

~~~
"Books a restaurant reservation on behalf of the agent.
 Hints: party_size = ['number of guests', 'for N people',
 'table for N']; datetime = ['next Tuesday', 'Saturday at 7']"
~~~

This enrichment improves MCP client accuracy when users express parameters in natural language rather than structured form.

(c) Tool inputSchema: use the AGIS input JSON Schema directly. Required parameters map to JSON Schema required array.

(d) Omit output, errors, confidence_guidance, and state_transition from the MCP tool definition. These have no MCP equivalent and MUST NOT cause generation failure.

Example conversion:

```
AGIS endpoint:
  BOOK /reservation {
    semantic {
      intent: "Books a restaurant reservation on behalf of the agent"
    }
    input {
      required: [restaurant_id, party_size, datetime]
      optional: [preferences]
    }
  }

Generated MCP tool entry:
  {
    "name": "book_reservation",
    "description": "Books a restaurant reservation on behalf of the agent",
    "inputSchema": {
      "type": "object",
      "properties": {
        "restaurant_id": { "type": "integer" },
        "party_size": { "type": "integer" },
        "datetime": { "type": "string", "format": "date-time" },
        "preferences": { "type": "string" }
      },
      "required": ["restaurant_id", "party_size", "datetime"]
    }
  }
```

C.3. MCP to AGIS Migration Path

Organizations with existing MCP tool catalogs can generate an initial AGIS document by applying the reverse mapping:

(a) Tool name -> method identifier: split on underscore, take the first segment, uppercase it, verify action-intent class. book_reservation -> BOOK (valid action-intent verb) get_data -> GET (invalid; HTTP method; rewrite as RETRIEVE or FETCH)

(b) Tool description -> intent statement: use verbatim if it describes agent goal; rewrite if it describes internal system operation.

(c) inputSchema -> input block: use directly.

(d) Add required semantic fields: actor (default: agent), outcome (derive from description or add manually), capability (classify manually from the five categories).

The resulting AGIS document will require manual review of semantic consistency (Pass 6) and MUST be validated using the agis-validator tool before publication.

C.4. References

C.5. Normative References

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

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

[JSON-SCHEMA] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", draft-bhutton-json-schema-01, Dec 2020.

[AGTP] Hood, C., "Agent Transfer Protocol (AGTP)", draft-hood-independent-agtp-03, 2026.

C.6. Informative References

[HOOD2026] Hood, C., "Semantic Method Naming and LLM Agent Accuracy: A Controlled Benchmark of REST/CRUD versus Agentive API Interface Design", Working Paper, Nomotic, Inc., March 2026. Available by request at chris@nomotic.ai.

[AGTP-METHODS] Hood, C., "AGTP Standard Extended Method Vocabulary", draft-hood-agtp-standard-methods-01, 2026.

[FIELDING] Fielding, R.T., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Dissertation, UC Irvine, 2000.

[FIPA-ACL] Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification", Document SC00061G, 2002. http://www.fipa.org/specs/fipa00061/

[FIPA-IP] Foundation for Intelligent Physical Agents, "FIPA Interaction Protocol Library Specification", Document SC00025G, 2002. http://www.fipa.org/specs/fipa00025/

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

[A2A] Linux Foundation, "Agent-to-Agent Protocol Specification", 2025. https://a2aprotocol.ai

Appendix D. Author's Address

Chris Hood
Nomotic, Inc.

Email: chris@nomotic.ai
URI: https://agtp.io
ORCID: [ORCID identifier]

Author's Address

Chris Hood
Nomotic, Inc.
Email: chris@nomotic.ai
URI: https://agtp.io