Independent Submission
Internet-Draft
Intended status: Informational
Expires: 27 November 2026

C. Hood
Nomotic, Inc.

26 May 2026

Agent Transfer Protocol (AGTP)
draft-hood-independent-agtp-08

Abstract

AI agents and agentic systems generate a growing volume of intent-driven, unstructured, and undifferentiated traffic that flows through HTTP indistinguishably from human-initiated requests. HTTP lacks the semantic vocabulary, observability primitives, and identity mechanisms required by agent systems operating at scale. Existing protocols described as Agent Group Messaging Protocols (AGMP), including MCP, ACP, A2A, and ANP, are messaging-layer constructs that presuppose HTTP as their transport. They do not address the underlying transport problem.

This document defines the Agent Transfer Protocol (AGTP): a dedicated application-layer protocol for AI agent traffic. AGTP is a runtime contract negotiation substrate (RCNS): a transport that fixes only a eighteen-method protocol floor and negotiates any additional method surface at runtime between agent and server in a single round-trip, governed by the AGTP-API companion specification [AGTP-API], which defines the curated method catalog, path grammar, endpoint primitive, and synthesis semantics. Version 07 confirms the IANA-registered agtp:// URI scheme and IANA-assigned port 4480 for TCP/TLS and QUIC, formalizes Form 1a URI grammar (agtp://{agent-id}@{host}) for direct addressing, renames the Agent Manifest Document to the Agent Identity Document with an enumerated schema, redesigns the protocol-defined method floor to a 12-method set organized as six cognitive verbs (QUERY, DISCOVER, DESCRIBE, SUMMARIZE, PLAN, PROPOSE) and six mechanics verbs (EXECUTE, DELEGATE, ESCALATE, CONFIRM, SUSPEND, NOTIFY), establishes AGTP as a substrate for higher-level agent frameworks (MCP, A2A, ACP) carried as content types inside AGTP method invocations, renumbers AGTP-specific status codes out of HTTP-assigned space to avoid semantic collision, mandates explicit Content-Length framing with a prohibition on TLS socket-level half-close, adds a .well-known/agtp bootstrap convention per RFC 8615, deprecates the AGIS reference and the proposed AGTP-Methods specification by folding both into the unified AGTP-API contract layer, adds status codes 405 (Method Not Allowed), 459 (Method Violation), and 460 (Endpoint Violation) per the AGTP-API contract model, and adopts "Agent Genesis" as the canonical term for the permanent signed origin document. Version 06 prepared the IANA

Service Name and Port Number application and consolidated the URI scheme registration. Version 05 restored the canonical Agent-ID as the primary identity primitive and decoupled Trust Tier 1 verification from DNS as a sole requirement. A canonical Agent-ID is derived from the agent's Agent Genesis hash and is authoritative in every AGTP protocol operation. Three equivalent verification paths are recognized for Trust Tier 1: DNS-anchored verification via RFC 8555 ACME challenge, log-anchored verification via Agent Genesis inclusion in an append-only transparency log aligned with RFC 9162 and RFC 9943 (SCITT), and hybrid verification combining DNS control with blockchain address ownership. Version 04 introduced normative integration hooks for the AGTP Merchant Identity and Agentic Commerce Binding specification [AGTP-MERCHANT], which defines the merchant-side identity model that complements AGTP's agent-side identity model. AGTP SHOULD prefer QUIC for new implementations and MUST support TCP/TLS for compatibility and fallback. It is designed to be composable with existing agent frameworks, not to replace them.

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 27 November 2026.

Copyright Notice

Table of Contents

1. Introduction

Note Regarding Intellectual Property: Implementers should be aware that extensions and certain mechanisms referenced in this document -- including the Agent Certificate extension (Section 7.2), the ACTIVATE method, the Agent Genesis mechanism (Section 5.7), and the .agent and .nomo file format specifications (Section 2) -- may be subject to pending patent applications by the author. The core AGTP specification is intended for open implementation without royalty obligation. The licensor is prepared to grant a royalty-free license to implementers consistent with [RFC8179]. IPR disclosures: https://datatracker.ietf.org/ipr/ -- see also Section 7.7.

1.1. Background

The deployment of AI agents and multi-agent systems is accelerating across enterprise, research, and consumer contexts. These systems execute complex, multi-step workflows, querying data sources, booking resources, delegating subtasks to peer agents, and escalating decisions to human principals, with minimal or no human supervision per transaction.

Unlike human-initiated web traffic, agent-generated traffic is dynamic, high-frequency, intent-driven, and often stateful across sequences of related requests. The infrastructure carrying this traffic was not designed with these properties in mind.

1.2. Limitations of HTTP for Agent Traffic

HTTP has served as the internet’s primary application-layer transport for over three decades. Its evolution through HTTP/2 [RFC7540] and HTTP/3 [RFC9114] has improved performance, multiplexing, and latency. However, the fundamental model of HTTP being stateless, resource-oriented, human-initiated request/response, creates specific failures when applied to agentic systems at scale:

* Traffic indistinguishability: Agent-generated requests are structurally identical to human-initiated requests at the transport layer. Operators cannot identify, route, or govern agent traffic without application-layer instrumentation.

* Method vocabulary mismatch: HTTP’s method set (GET, POST, PUT, DELETE, PATCH) describes resource operations. Agent traffic expresses purposeful intent, summarize, book, delegate, escalate. The mismatch forces intent into request bodies, invisible to protocol-level handlers.

* Identity and attribution absence: HTTP carries no native mechanism for asserting agent identity, declared authority scope, or the principal accountable for an agent's actions.

* Session semantics mismatch: HTTP's stateless model is optimized for isolated request/response cycles. Agent workflows are inherently stateful sequences.

1.3. Why Not Evolve HTTP?

A natural question is whether these limitations could be addressed by extending HTTP rather than defining a new protocol. There are three specific reasons why HTTP extension is not the preferred path.

First, the HTTP method registry is effectively frozen for new semantics. [RFC9110] defines the HTTP method registry with IETF Review as the registration procedure, meaning new methods require a full IETF consensus process and must be backward-compatible with existing HTTP implementations. Adding intent-based verbs (SUMMARIZE, DELEGATE, ESCALATE) to HTTP would require every HTTP client, server, proxy, and middleware component to ignore or handle unknown methods gracefully, a compatibility constraint that limits how agent-specific semantics can be expressed at the protocol level.

Second, HTTP carries decades of backward-compatibility constraints. Features such as persistent agent identity headers, authority scope declarations, and session-level governance semantics would require HTTP extensions that interact unpredictably with existing caching, proxy, and CDN behavior designed for human-generated traffic patterns.

Third, the observability goal making agent traffic distinguishable from human traffic at the infrastructure layer cannot be achieved by adding fields to HTTP. Infrastructure components route and filter HTTP traffic based on methods and headers that are identical across agent and human requests. A protocol-level separation is necessary to give infrastructure the signal it needs.

AGTP is therefore designed as a dedicated protocol rather than an HTTP extension. HTTP and AGTP coexist: human traffic continues to flow over HTTP; agent traffic flows over AGTP. The two protocols serve different classes of network participant.

Note: The abbreviation AGTP is used in this document to distinguish the Agent Transfer Protocol from the Authenticated Transfer Protocol (ATP) working group currently chartered within the IETF. The URI agtp:// is proposed for IANA registration as a new and distinct scheme.

1.4. Motivation for a Dedicated Protocol

These limitations are architectural, not implementational. They cannot be resolved by better middleware or application code layered on HTTP. They require a protocol designed from first principles for AI agent systems.

AGTP is that protocol. It provides a dedicated transport environment for agent traffic with: native intent-based methods, mandatory agent identity headers, protocol-level authority scope declaration, and a status code vocabulary for the conditions AI systems encounter.

1.5. Scope and Target Audience

This document covers AGTP architecture, design principles, stack position, request and response header format, agent-native method definitions and semantics, status code vocabulary, security considerations, and IANA considerations.

The Agent Certificate extension for cryptographic binding of agent identity to AGTP header fields is described at a high level in Section 7.2. Full specification is provided in a separate companion document: [AGTP-CERT]. That extension may be subject to pending intellectual property claims; see Section 7.7 and the IPR Notice preceding the Abstract.

Merchant-side identity verification for PURCHASE counterparties is described at a high level in Section 8 of this document and specified in full in a separate companion: [AGTP-MERCHANT]. This document registers the merchant-related request headers, the 458 Counterparty Unverified status code, and the merchant and intent Authority-Scope domains; the Merchant Manifest Document, Merchant Agent Genesis, counterparty verification procedure, and Intent Assertion JWT format are specified in the companion.

Target audience: AI agent developers, protocol designers, cloud and network infrastructure providers, enterprise security and compliance architects, and standards community participants.

1.6. AGTP as the Transport Foundation for Agent Group Messaging Protocols

AGTP is the purpose-built transport and governance layer for Agent Group Messaging Protocols (AGMPs): the category of higher-layer AI agent messaging standards that includes the Model Context Protocol (MCP) [MCP], the Agent-to-Agent Protocol (A2A) [A2A], the Agent Communication Protocol (ACP) [ACP], and emerging others.

AGMPs define what agents say. AGTP defines how those messages move, who sent them, and under what authority. AGTP provides the narrow-waist foundation that AGMPs inherit without modification: intent-native methods, mandatory agent identity and scoping, resource budget enforcement, observability hooks, and normative composition profiles. A deployment running any AGMP over AGTP gains transport-level governance without changes to the messaging layer.

The AGMP category term is introduced in this document to provide a stable collective reference for the class of protocols that AGTP serves as substrate. It is not a formal IETF term of art; it is a descriptive classification. Individual AGMP specifications retain their own names and development paths. AGTP does not govern, modify, or supersede any AGMP.

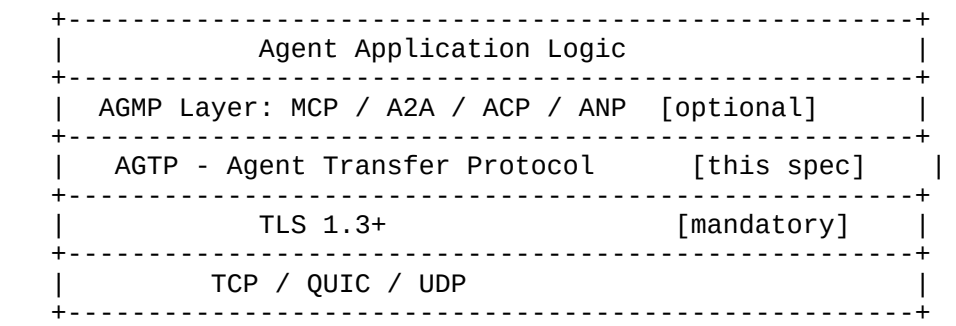

Figure 1: AGTP as Substrate for AGMPs

2. Terminology

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

Agent: An AI software system that executes tasks, makes decisions, and takes actions without continuous human supervision per transaction.

Principal: The human, organization, or system that authorized an agent to act and is accountable for its actions.

Agent-ID: A unique identifier for a specific agent instance. Carried in the Agent-ID request header on non-anonymous AGTP requests, and in the agent_id field of the Agent Identity Document.

Principal-ID: The identifier of the principal on whose behalf an agent operates. Carried in the agent identity document referenced by Agent-ID; not transmitted as a separate request header.

Authority-Scope: A declared set of permissions defining what actions an agent is authorized to take, in the format domain:action or domain:*. Declared in the agent's identity document. *MAY* be carried on individual requests as a claimed-scopes header narrowing the agent's full authorized set to those needed for the request; claimed scopes *MUST* be a subset of the document's declared set.

Intent Method: An AGTP method name expressing the agent's purpose, as distinguished from HTTP resource-operation verbs.

Delegation Chain: An ordered record of Agent-IDs representing the sequence of delegations that produced the current request.

Escalation: An agent's intentional deferral of a decision or action to a human principal or higher-authority agent.

Attribution Record: A logged record of an agent action sufficient for audit and compliance purposes.

Session: An AGTP persistent connection context shared across multiple method invocations within a single agent workflow.

SEP (Scope-Enforcement Point): An AGTP-aware infrastructure component, load balancer, gateway, and proxy, that enforces Authority-Scope compliance without application-layer access. Requires the Agent Certificate extension ([AGTP-CERT]).

Agent Package (.agent): A portable, open deployment artifact for an AI agent. An .agent file contains an embedded Agent Manifest, an integrity hash covering all package contents, and a behavioral trust score computed at packaging time. The .agent format is an open specification. It is analogous to a container image: a self-describing, portable unit of deployment. The .agent suffix is a file format designator and *MUST NOT* appear as a hostname component or top-level label in agtp:// URIs. Note: the .agent file format specification may be subject to pending patent claims by the author; see Section 7.7.

Governed Agent Package (.nomo): A deployment artifact in the .nomo format, which extends the .agent format with a CA-signed certificate chain binding the package to a verified governance zone and issuing principal. The .nomo format is to .agent as HTTPS is to HTTP: the same structural foundation with an added

layer of cryptographic trust. A .nomo package is required for agents operating at Trust Tier 1 (see Section 5.2). The .nomo suffix is a file format designator and *MUST NOT* appear as a hostname component in agtp:// URIs.

The name derives from the Greek _nomos_ ($\nu \mu o$), meaning law, rule, or governance, the same root that underlies _autonomy_ (self-law), _nomocracy_ (rule of law), and _onomastics_. A .nomo package is literally an agent operating under law: its behavior is bounded by a cryptographically enforced governance context at the packaging layer. Note: the .nomo file format specification may be subject to pending patent claims by the author; see Section 7.7.

Agent Transfer Document (.agtp): The wire-level manifest document format defined by this specification. An .agtp document is a signed JSON structure containing the fields defined in Section 5.5 (Agent Identity Document). It is the output format returned by all AGTP URI resolution requests. Both .agent and .nomo packages produce .agtp documents when queried; the .agtp format is the protocol's canonical representation of agent identity and is independent of the underlying packaging format. The .agtp suffix *MAY* appear in filenames for stored manifest documents but *MUST NOT* appear in agtp:// URIs. The Content-Type for .agtp documents is application/vnd.agtp+json.

URI (AGTP): An agtp:// scheme URI that identifies an agent or agent namespace. AGTP URIs are addresses, not filenames. File extensions (.agent, .nomo, .agtp) *MUST NOT* appear in canonical AGTP URIs. See Section 5.1 for the canonical URI forms and resolution semantics.

Agent Namespace Document: A cryptographically signed application/vnd.agtp+json document returned in response to a request targeting an organization's agent registry root (e.g., agtp://acme.tld/agents). Lists all Active agents registered under the organization's governance zone. The document is generated and re-signed by the governance platform on any registry change. It is not a manually editable file. See Section 5.4.

Agent Identity Document: A cryptographically signed application/vnd.agtp+json document returned in response to a request targeting a specific agent (e.g., agtp://acme.tld/agents/customer-service). Contains the agent's Agent Genesis fields, lifecycle state, behavioral trust score, authority scope categories, supported methods, and governance zone. Derived directly from the agent's .agent or .nomo package; the package integrity hash is verified before the manifest is served. See Section 5.5.

Agent Genesis: A cryptographically signed origin document issued to an agent at registration time by a governance platform. The Agent Genesis is the genesis record of an agent's existence: it establishes the agent's identity, ownership, authorized scope, behavioral archetype, and governance zone before the agent takes any action. Authority is issued through the Agent Genesis; it is never self-assumed.

The Agent Genesis is the source document from which the Agent Identity Document (Section 6.4) is derived when an AGTP URI is resolved. The canonical Agent-ID is computed as the 256-bit SHA-256 hash of the Agent Genesis in its canonical-form serialization with the signature field excluded; the canonical form is specified in Section 6.5.2. In this sense the Agent Genesis functions as the agent's permanent origin record: issued once at creation, permanently bound to the agent, and the authoritative identity record from which all other identity representations derive.

Agent Genesis fields map to AGTP protocol elements: agent_id maps to the Agent-ID header on every request; owner is the principal identifier recorded in the agent identity document referenced by Agent-ID (not a separate header); scope is the Authority-Scope set declared in the agent identity document, optionally narrowed per-request via the Authority-Scope header. See Section 5.7.

Anonymous agents are ungovernable. Without an Agent Genesis, there is no mechanism to trace decisions to a responsible principal, enforce scope boundaries, or maintain a meaningful audit trail.

The taxonomy is: *Agent Genesis* (the permanent signed governance-layer origin document) → *canonical Agent-ID* (the 256-bit hash of the Agent Genesis, used in all AGTP protocol operations) → *Agent Certificate* (an optional X.509 v3 credential for TLS mutual authentication; specified in [AGTP-CERT]). Note: the Agent Genesis mechanism may be subject to pending patent claims by the author; see Section 7.7.

Governance Token: A signed, time-limited JWT artifact issued by a

governance runtime that encodes a specific governance decision for a specific action. Governance tokens are the runtime companion to the static Agent Genesis: where the Agent Genesis establishes persistent identity, the Governance Token carries a bounded authorization for a single action or session. Tokens carry the governance verdict (ALLOW, DENY), the agent ID, action details, trust score dimensions, issuer identity, and expiry. Default TTL: 30 seconds. Tokens *MUST NOT* be reused across actions; each action requires a fresh evaluation and a fresh token.

Trust Tier: A classification assigned to an agent based on the strength of identity verification backing its registration. Tier 1 (Verified): org anchor is a real DNS domain with confirmed ownership and a .nomo governed package. Tier 2 (Org-Asserted): org label is present but DNS ownership is unverified; .agent package acceptable. Tier 3 (Experimental): X- prefix required; not discoverable through the public AGTP registry. See Section 5.2.

AGMP (Agent Group Messaging Protocol): The collective term for higher-layer AI agent messaging standards that operate over AGTP as their transport substrate, including MCP [MCP], A2A [A2A], ACP [ACP], and ANP [ANP]. AGMPs define what agents say to each other. AGTP defines how those messages move. The term is introduced in this document as a descriptive classification; it is not a formal IETF term of art.

DESCRIBE: An AGTP cognitive floor method that returns the declared capabilities, supported modalities, method vocabulary, and versioned feature set of a specific agent endpoint. Distinguished from URI resolution (which returns identity) by returning operational capability metadata suitable for pre-task negotiation. If the capability_domains parameter is omitted, the server *SHOULD* return all supported domains. Category: ACQUIRE.

SUSPEND (method): An AGTP mechanics floor method that places a specific active session workflow into a recoverable paused state, issuing a resumption nonce for re-entry. Distinguished from the lifecycle SUSPEND event (Section 6.7.6): method-level SUSPEND is session-scoped and does not affect the agent’s registry lifecycle state or Agent Genesis validity. Category: ORCHESTRATE.

Budget-Limit: A request header declaring the maximum resource consumption the principal authorizes for a method invocation, expressed as comma-separated unit=value tokens drawn from the IANA AGTP Budget Unit Registry per [RFC9110] list-valued header conventions. Example: Budget-Limit: tokens=5000, compute-seconds=120, financial=10.00USD, ttl=3600. Exceeding the declared

limit *MUST* cause the server to return 456 Budget Exceeded rather than continue execution. Note: ttl= is RECOMMENDED to bound budget lifetime. Reserved for v01+ per Section 5.5.

AGTP-Zone-ID: A request header declaring the network zone or organizational boundary within which a request must be processed. Scope-Enforcement Points (SEPs) *MUST* enforce zone boundaries and *MUST* return 457 Zone Violation if a DELEGATE request would route outside the declared zone.

3. Problem Statement

AGTP is motivated by three distinct, compounding failures in how current internet infrastructure handles AI agent traffic.

3.1. Problem 1: Undifferentiated Agent Traffic on HTTP

AI agents generate intent-driven, structured traffic that is functionally invisible to the infrastructure it traverses. This traffic flows through HTTP alongside human traffic with no protocol-level differentiation. Observability failure, routing inefficiency, and security blindness result, operators cannot determine what fraction of traffic is agent-generated without application-layer instrumentation that is expensive, inconsistent, and easy to circumvent.

AGTP response: a dedicated protocol environment for agent traffic. Infrastructure can distinguish, route, monitor, and govern agent traffic natively.

3.2. Problem 2: Semantic Mismatch Between Agent Intent and Available Methods

AI agents operate on intent. HTTP’s method vocabulary was designed to describe operations on resources, not purposeful action. When an agent intends to SUMMARIZE a document, EXECUTE a reservation, and PLAN a sequence, all three arrive as POST requests. The server receives identical verbs with meaningfully different intent buried in request bodies, invisible to any protocol-level handler.

AGTP response: a vocabulary of agent-native methods that express intent at the protocol level.

3.3. Problem 3: No Protocol-Level Identity, Authority, or Attribution for Agents

When an AI agent takes an action, there is currently no protocol-level mechanism to verify who authorized this agent, what scope of authority it holds, which principal is accountable for its actions, or whether it is the agent it claims to be. Accountability gaps, authority laundering, auditability failure, and multi-agent trust collapse result.

AGTP response: agent identity and authority scope embedded in protocol headers on every request, with an optional Agent Certificate extension for cryptographic verification.

3.4. Problem Summary

#	Problem	Current Failure	AGTP Response
1	Undifferentiated traffic	HTTP cannot separate agent traffic	Dedicated protocol environment
2	Semantic mismatch	HTTP verbs obscure agent intent	Native intent-based method vocabulary
3	No protocol-level identity	Attribution is untraceable	Agent identity and scope in headers

Table 1: Summary of Problems Addressed by AGTP

4. Related Work and Existing Approaches

4.1. HTTP/REST as the De Facto Standard

HTTP remains the universal transport for all agent traffic currently deployed. REST conventions layered on HTTP provide a degree of semantic structure, but REST remains a resource-manipulation paradigm. As described in Section 1.3, evolving HTTP to address agent-specific needs is constrained by the frozen method registry, backward-compatibility requirements, and the impossibility of achieving infrastructure-level traffic differentiation through HTTP extensions alone.

4.2. Existing Agent Group Messaging Protocols

MCP [MCP] (Model Context Protocol, Anthropic): Defines structured

communication between AI models and tools/resources. Runs over HTTP. Addresses tool-calling semantics, not agent traffic transport.

ACP [ACP] (Agent Communication Protocol, IBM): Defines messaging semantics for agent-to-agent communication. Runs over HTTP.

A2A [A2A] (Agent-to-Agent Protocol, Linux Foundation): Defines inter-agent communication and task delegation semantics. Runs over HTTP.

ANP [ANP] (Agent Network Protocol): Defines discovery and communication for networked agents. Runs over HTTP.

All of these are messaging protocols. They define what agents say to each other. They do not define how agent traffic moves across a network. Each presupposes HTTP as its transport and inherits all of HTTP's limitations for agentic systems.

4.3. Transport-Layer Alternatives

gRPC: High-performance RPC over HTTP/2. Strong typing and efficient serialization. Does not address agent-specific semantics, identity, or authority.

WebSockets: Persistent bidirectional connections over HTTP. Useful for real-time communication but does not address method semantics or identity.

QUIC [RFC9000]: Modern multiplexed transport with reduced connection overhead. AGTP *SHOULD* prefer QUIC for new implementations. QUIC is a transport primitive; AGTP is the application-layer protocol above it.

4.4. The Critical Distinction: Messaging vs. Transport

The most important positioning principle for AGTP is the distinction between messaging protocols and transport protocols. MCP, ACP, A2A, and ANP are messaging protocols, they define what agents say. AGTP defines how agent traffic moves.

An analogy: SMTP is a messaging protocol that runs over TCP. SMTP does not replace TCP. Saying "TCP is unnecessary because SMTP exists" is a category error. The same logic applies here. MCP and its peers define agent messaging semantics. AGTP defines the transport environment those messages move through.

4.5. AGTP Positioning: The Proposed Stack

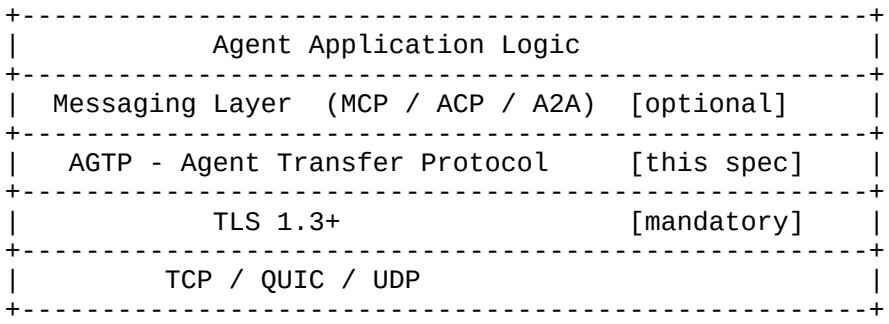

Figure 2: AGTP in the Protocol Stack

AGTP is not a replacement for messaging protocols. Agents using MCP or A2A route those messages over AGTP and gain transport-level observability and identity without modifying the messaging layer. AGTP-native agents that do not use a separate messaging protocol interact with AGTP methods directly.

5. Protocol Overview

5.1. Stack Position

AGTP is an application-layer protocol. It operates above the transport layer (TCP, UDP, or QUIC) and is wrapped by TLS. It sits below any agent messaging protocol in deployments that use one.

* *SHOULD* prefer QUIC [RFC9000] [RFC9001] for new deployments (lower latency, multiplexing without head-of-line blocking, 0-RTT connection establishment).

* *MUST* support TCP/TLS as a fallback for compatibility with existing infrastructure.

* *MAY* run over UDP where QUIC is not available, subject to implementor-defined reliability guarantees.

AGTP uses port *4480* (TCP and UDP), assigned by IANA under the service names agtp (TCP/TLS) and agtp-quic (QUIC). The port assignment is permanent and applies to all AGTP transports. Full IANA registration metadata is documented in Section 11.1.

5.2. Design Principles

Minimalist core: The base spec defines only what is necessary for

agent traffic differentiation, method semantics, and identity headers. Extensions belong in companion specifications.

Extensible by design: New methods are registered through an IANA-managed Method Registry. New header fields follow a defined extension convention. Additive changes do not require a version increment.

Agent-native: Every design decision assumes the initiating party is an AI system, not a human.

Secure by default: TLS 1.3 or higher is mandatory. Unencrypted AGTP connections *MUST* be rejected. Agent identity headers are present on every request.

Observable by design: Native metadata in every AGTP header provides the minimum information needed for routing, monitoring, and audit without application-layer instrumentation.

Composable: AGTP works alongside existing agent messaging protocols without requiring modification to those protocols.

Runtime contract negotiation: AGTP fixes the protocol surface at eighteen methods. Beyond that floor, the surface is not predetermined. An agent that needs an endpoint the server does not advertise proposes the endpoint via PROPOSE with an AGTP-API endpoint definition; the server evaluates whether it can synthesize the endpoint from existing capabilities and either instantiates it as a session-scoped endpoint or refuses. The negotiation completes in a single round-trip. See Section 5.3.

5.3. AGTP as a Runtime Contract Negotiation Substrate

AGTP is a runtime contract negotiation substrate (RCNS). The substrate fixes a small floor of eighteen protocol-level methods (the cognitive and mechanics verbs of Section 7.1); beyond that floor, the method and endpoint surface that any given AGTP server presents is not fixed in advance. It is negotiated at runtime between the agent and the server, in a single round-trip, governed by the AGTP-API companion specification [AGTP-API] which defines the curated method catalog, path grammar, endpoint primitive, semantic block, schemas, and synthesis semantics.

The negotiation loop is:

1. An agent encounters an AGTP server and inspects its endpoint surface via DESCRIBE or by retrieving the server manifest as defined in [AGTP-API].

2. If the endpoints the agent needs are not present, the agent submits a PROPOSE request carrying an AGTP-API endpoint specification: a verb (drawn from the AGTP-API approved verb list), a path (conforming to AGTP-API path grammar), a semantic block (intent, actor, outcome, capability classification, confidence guidance, impact tier, idempotency), input and output schemas, and declared error conditions.

3. The server evaluates the proposal against AGTP-API contract rules and against its own capability surface. The server determines whether the proposed endpoint can be synthesized from existing endpoints, what authority scope is required, and what governance constraints apply.

4. The server either instantiates the proposed endpoint as a session-scoped endpoint and returns 263 Proposal Approved with the AGTP-API endpoint definition, or refuses with 463 Proposal Rejected and a structured reason.

The full negotiation completes in a single PROPOSE → response round-trip. No registry update, no human-in-the-loop approval, no out-of-band coordination is required for an agent and a server to agree on a new endpoint, provided the endpoint is AGTP-API conformant and within the server’s capability envelope.

This property distinguishes AGTP from protocols whose contract surface is fixed at design time. HTTP’s method registry is effectively frozen (see Appendix E); proposing a new HTTP method requires IETF consensus over multi-year timelines. gRPC services expose a fixed service contract defined at compile time. Most agent frameworks built on HTTP inherit this constraint and work around it by overloading POST. AGTP treats the contract surface as dynamically negotiable, with AGTP-API as the contract layer that keeps negotiation safe.

The RCNS property is what makes composition with higher-level agent frameworks (Section 8) tractable. A server that wants to expose a framework-specific operation as a first-class AGTP endpoint does not need to wait for IANA registration; it negotiates the endpoint via PROPOSE at the moment an agent asks for it, governed by AGTP-API. The floor of eighteen methods provides interoperability; AGTP-API provides contract safety; PROPOSE provides expressivity.

Implementations *MAY* choose not to participate in runtime negotiation. A server that supports only the eighteen-method floor and returns 463 Proposal Rejected for every PROPOSE request is fully conformant with this specification. Runtime negotiation is a capability AGTP enables, not a behavior it requires.

5.4. Connection Model

AGTP uses a persistent session model by default, reflecting the reality that agents typically execute multi-step workflows rather than isolated single requests. An AGTP session is established with a single TLS handshake including agent identity assertion, persists across multiple method exchanges, carries a Session-ID header identifying the agent's task context, and terminates on explicit session close or inactivity timeout (RECOMMENDED minimum: 60 seconds).

Per-request (stateless) mode is supported for constrained environments. In stateless mode, agent identity headers *MUST* be present on every individual request.

5.4.1. Wire-Format Framing

AGTP requests and responses *MUST* be framed by an explicit Content-Length header. Content-Length is the sole signal of request and response completion. Receivers *MUST* treat the message as complete when, and only when, the declared number of body octets has been read after the header terminator.

AGTP sessions running over TLS *MUST NOT* use socket-level half-close (shutdown(SHUT_WR) or equivalent) to signal end-of-request. The TLS close_notify alert that results from a half-close terminates the secure session before the peer can transmit a response, producing a truncation that is indistinguishable at the application layer from a malicious downgrade. Implementations that require an explicit completion signal in addition to Content-Length *MUST* rely on the AGTP session-close semantics described in Section 7.2.12 or on transport-level FIN after the full response has been received.

Chunked transfer encoding is not used in AGTP. Streaming method responses are framed by repeated Content-Length-delimited messages within a single AGTP session.

5.5. Header Format

The AGTP wire format puts the protocol version and method on the request line (AGTP/1.0 METHOD PATH\r\n) and the status code on the response line (AGTP/1.0 STATUS STATUS-TEXT\r\n). Headers *MUST NOT* carry information already present on those lines: there is no AGTP-Version header, no AGTP-Method header, no AGTP-Status header. The version and method are read from the request line; the status is read from the response line.

5.5.1. Request Line Grammar

The request line carries three tokens separated by single space characters, terminated by CRLF:

```
request-line   = agtp-version SP method SP request-target CRLF
agtp-version   = "AGTP/1.0"
method         = token                  ; per {{AGTP-API}} catalog
request-target = path-absolute [ "?" query ]
                                        ; per RFC 3986
path-absolute  = "/" [ segment-nz *( "/" segment ) ]
                                        ; per RFC 3986 Section 3.3
query          = *( pchar / "/" / "?" )
                                        ; per RFC 3986 Section 3.4
```

The request-target follows the path-absolute production of [RFC3986], optionally followed by ? and a query string. Path and query are parsed as separate tokens: the path is everything from the leading / up to the first ? or end-of-line; the query string is everything after the first ? up to the CRLF.

Servers *MUST* parse path and query as separate tokens before dispatch, matching [RFC3986] URI generic syntax. Implementations *MUST NOT* treat a ?-prefixed query as part of the path; the path-grammar enforcement in [AGTP-API] applies to the path component only.

URI fragments (#anchor) *MUST NOT* appear on the request line. A request line containing # *MUST* be rejected at the wire layer as malformed.

The path-pattern grammar (template parameters in {param} form, verb-leakage prevention, structural minimums) is normatively specified in [AGTP-API]. v07 servers *MUST* implement the AGTP-API path grammar.

5.5.2. Response Line Grammar

The response line carries three tokens separated by single space characters, terminated by CRLF:

```
response-line  = agtp-version SP status-code SP status-text CRLF
agtp-version   = "AGTP/1.0"
status-code    = 3DIGIT                 ; per {{status-codes}}
status-text    = *( VCHAR / SP / HTAB )
                                        ; freeform reason phrase
```

The status-text is informational and *MUST NOT* be used by servers or clients for protocol decisions; the numeric status-code is authoritative.

5.5.3. Request Headers

Field	Required	Description
Agent-ID	*MUST* for non-anonymous requests	Canonical 256-bit identifier of the invoking agent. Identifies the request's source, not its target. Servers reject requests lacking Agent-ID against non-anonymous endpoints.
Authority-Scope	*MAY*	Scopes the agent claims for this specific request. When present, every claimed scope *MUST* be a subset of the scopes declared in the agent's identity document; servers *MUST* validate and *MUST* return 262 Authorization Required with body code scope-claim-invalid on failure. When absent, the request inherits the full scope set of the agent's identity document. The server uses the claimed (or inherited) set when evaluating the endpoint's required_scopes.
Session-ID	*MAY*	Opaque session identifier grouping multiple requests into an operational session. The protocol assigns no semantics; servers pass the value

		through to handlers via the endpoint context.
Task-ID	*MAY*	Opaque identifier tracing a specific task or operation across multiple requests. Useful for audit correlation. Servers *SHOULD* echo this value in the response Task-ID header.
Delegation-Chain	*MAY* (reserved)	Reserved for delegated-authority scenarios. Format, validation, and chain-of-trust semantics are not specified in this revision and are anticipated in a future revision. v00 servers *MAY* reject requests carrying this header with 501 Not Implemented (HTTP standard semantics); reject responses *SHOULD* carry a body indicating the unsupported feature.
Merchant-ID	*MUST* on PURCHASE	Canonical identifier of the intended merchant counterparty. See [AGTP-MERCHANT].
Merchant-Manifest-Fingerprint	*MUST* on PURCHASE	SHA-256 fingerprint of the Merchant Manifest Document verified by the requesting agent. Receiving server *MUST* reject with 458 if this does not match its current manifest. See [AGTP-MERCHANT].

Intent-Assertion	*SHOULD* on PURCHASE	Detached JWT [RFC7519] carrying signed principal-authorized purchase intent. Forwardable to payment networks as standalone evidence. See [AGTP-MERCHANT].
Authorization	*MAY*	Application-layer authorization credential carried for composition with external identity providers (e.g., OIDC/ OAuth bearer tokens, SPIFFE SVIDs). When present, the value follows the HTTP Authorization header syntax per [RFC9110] (e.g., Bearer TOKEN). AGTP identity (Agent-ID, Agent Certificate) and the credential in this header answer different questions and are orthogonal: AGTP identity identifies the requesting agent; the credential in this header identifies the human or service principal on whose behalf the agent acts. Servers *MAY* require an Authorization header on specific methods per [policies.oauth] configuration (Section 8.6); semantics of any token in the header are application-defined and outside AGTP’s

		wire scope.
Cart-Digest	*MAY*	Cryptographic digest of a structured cart returned by a prior QUOTE invocation. Binds a PURCHASE to a previously quoted cart without retransmission of line-item detail. See [AGTP-MERCHANT].
Allow-RCNS	*MAY*	Caller opt-in to Runtime Contract Negotiation. Value true signals the caller is willing to receive a 461 RCNS Contract Available response or an inline optimistic synthesis when a requested (method, path) pair is unregistered. Absent or non-true keeps the RCNS gate closed; the request is then refused per the standard unregistered-pair rules. See [AGTP-API].
Contract-Synthesized	*MAY*	The synthesis_id of a previously-issued RCNS contract. Presents the contract to bypass the four-lock gate and dispatch the request directly under the synthesized contract. The server *MUST* refuse the presentation with 464 contract-not-yours if the caller’s Agent-ID does not match the contract’s originating_agent_id.

		See [AGTP-API].
Idempotency-Key	*MAY*	Per-agent idempotency key for RCNS-eligible requests and other state-modifying calls. Servers *MUST* scope the idempotency cache by (Agent-ID, Idempotency-Key); replays from the same agent return the cached response, replays from different agents are independent. See [AGTP-API].

Table 2: AGTP Request Header Fields

5.5.4. Response Headers

Field	Required	Description
Server-ID	*MUST*	Canonical identifier of the server that produced the response. *MUST* be populated from the server's configured server_id on every response. Useful for audit, load-balanced deployments, and verifying which server processed a request. Distinct from Agent-ID: Server-ID identifies the host that processed the request, Agent-ID (when echoed) identifies the requesting agent.
Agent-ID	*SHOULD* when present in request	Echo of the request's Agent-ID header to correlate response with the requesting agent. Absent when the request was anonymous. Echoed verbatim from the request; servers *MUST NOT* substitute or normalize the value on the response path.
Task-ID	*SHOULD* when present in request	Echo of the request's Task-ID header to correlate response with request. Absent when the request did not carry Task-ID.
Request-ID	*SHOULD*	Echo of the request's Request-ID header per the per-intera

```
ction       |
   |            |when present|identifier convention.  See [AGTP-IDENTIFIERS].
            |
   |            |in request  |
            |
   +-------------+------------+-------------------------------------------------------
------------+
   |Response-ID  |*MUST*      |Per-interaction identifier of this response.  Distinct fro
m Request-  |
   |             |            |ID; together they pair a request and its response on the w
ire.  See   |
```

		|[AGTP-IDENTIFIERS].
Audit-ID	*MUST* when Attribution-Record is present	Identifier of this response's Attribution-Record, computed as sha256(jws_compact_serialization) of the Attribution-Record JWS. The same value appears as previous_audit_id in the agent's next emitted Attribution-Record, closing the per-agent hash chain. See [AGTP-IDENTIFIERS].
Owner-ID	*SHOULD*	Identifier of the human or organizational principal accountable for the responding agent. Recorded in the agent's Agent Genesis and stamped on every response by the daemon. See [AGTP-IDENTIFIERS].
Trust-Tier	*SHOULD*	Resolved trust tier of the responding agent (1, 2, or 3) per the trust-posture loading rule in [AGTP-TRUST]. Stamped on every response when the responding server has resolved a trust tier for the addressed agent; omitted when no trust tier is known. Allows relying parties to apply trust-tier-conditional policy on every response without consulting the Agent Identity Document.
Verification-Path	*SHOULD* when Trust-Tier is present	Resolved verification path of the responding agent (dns-anchored, log-anchored, hybrid, or org-asserted) per the trust-posture loading rule in [AGTP-TRUST]. Stamped on every response when the responding server has resolved a verification path; omitted when no path is known.
Trust-Warning	*MUST* when resolved Trust-Tier is 2 and a warning is set	Trust warning token recorded on the responding agent's Agent Identity Document (e.g., verification-incomplete, verification-path-unsupported). Stamped on every response from a Tier 2 agent so relying parties can surface the warning without consulting the Agent Identity Document; omitted on Tier 1 and Tier 3 responses. See [AGTP-TRUST].

Contract-Synthesized-Untime-	*MUST* when serving under RCNS optimistic mode	The synthesis_id of the contract under which this response was served. Notifies the caller that the response was served under a r negotiated contract per [AGTP-API]. Callers that wish to repeat the action send subsequent requests with the same header value in the request to bypass the gate.
RCNS-Attempt-Retrievable-Id	*MUST* on every 464 RCNS No Contract response from an RCNS-capable server	Opaque identifier of the failed synthesis attempt record. via INSPECT target=rcns-attempt and carries the diagnostic detail for the failure (which gate lock closed, the requested (method , path), the trust tier resolved at evaluation time). See [AGTP-API].
Attribution-Record	*MUST*	JWS-signed attestation of the response’s origin, serialized in JWS Compact form per [RFC7515]

			(base64url(protected_header).base64url(payload).base64url(signature)).
			The payload carries the base attribution fields (server ID, response
			timestamp, request hash, response status) together with the
			identifier-chain fields and the per-agent chain link
			(previous_audit_id); see [AGTP-IDENTIFIERS] for the extended payload
			schema. Servers with a configured manifest signing key sign the
			payload with that key; servers without a configured key emit a
			fallback JWS with alg: none and an empty signature octet, preserving
			the wire format and the Audit-ID chain. Consumers that require
			cryptographic attestation *MUST* reject alg: none records.
Continuation-Token	*MAY*	Token for retrieving additional results in streaming contexts.
Supported-Methods	*SHOULD* (on session open)	List of AGTP methods supported by this server.
Cost-Estimate	*MAY*	Estimated resource consumption in Budget-Limit unit format. Returned by QUOTE; *MAY* appear on any response as an informational signal.
Attestation-Evidence	*MAY*	RATS attestation evidence token or reference URI per [RFC9334]. Format indicated by attestation_type in response body: rats-eat, rats-corim, or rats-uri.

Table 3: AGTP Response Header Fields

Implementations *MAY* emit implementation-specific headers, by convention prefixed X-, but such headers have no protocol semantics; agents *MUST NOT* rely on them and servers *MUST NOT* require them.

5.5.5. Headers Reserved for Future Revisions

The following headers were specified in earlier drafts and are reserved for future revisions. v00 servers *MUST NOT* require them and *MAY* ignore them when received:

Field	Status	Anticipated Use
Priority	Reserved (v01+)	Request priority hint: critical, normal, background
TTL	Reserved (v01+)	Maximum acceptable response latency in milliseconds. Pairs with status code 408 Timeout.
Budget-Limit	Reserved (v01+)	Maximum resource budget per invocation. Pairs with status code 456 Budget Exceeded.
AGTP-Zone-ID	Reserved (v01+)	Network zone boundary constraint. Pairs with status code 457 Zone Violation; SEP-enforced.
Content-Schema	Reserved (v01+)	URI reference to JSON Schema describing the request body structure.
Telemetry-Export	Reserved (v01+)	OTLP endpoint URI for metric export, or inline to receive metrics embedded in the response Attribution-Record.

Table 4: Headers Reserved for Future AGTP Revisions

Status codes 456 Budget Exceeded and 457 Zone Violation remain allocated in the AGTP Status Code Registry; their normative trigger headers (Budget-Limit, AGTP-Zone-ID) are deferred to a future revision. v00 servers *MAY* enforce equivalent semantics through implementation-specific mechanisms and return the corresponding status code, but the wire-level headers are not part of the v00 contract.

5.5.6. Retired Headers

The following headers appeared in earlier drafts and have been removed. Implementations *MUST NOT* emit them and *MAY* reject requests that carry them:

Field	Reason for Removal
AGTP-Version	Redundant with the request and response lines (which carry AGTP/1.0).
AGTP-Method	Redundant with the request line (which carries the method token).
AGTP-Status	Redundant with the response line (which carries the status code).
Principal-ID	Redundant: the principal is recorded in the agent identity document referenced by Agent-ID.
Server-Agent-ID	Renamed to Server-ID. The earlier name conflated server identity with agent identity.

Table 5: Retired AGTP Headers

5.6. Status Codes

AGTP defines its own status code space. Codes 261 through 263, 455 through 465, and 550 through 555 are AGTP-specific with no HTTP equivalent and are registered in the IANA AGTP Status Code Registry (see Section 9.3). AGTP-specific code numbers are deliberately chosen from ranges unassigned in the IANA HTTP Status Code Registry to avoid semantic collision with HTTP status codes that may appear in payloads carried by AGTP method invocations.

The AGTP status code model carries four structural rejection codes (404, 405, 459, 460) that together cover the failure surface for contract-level invocation. Each code is independently actionable: 404 indicates the path does not exist on this server; 405 indicates the verb and path are each individually valid but the server does not expose this combination; 459 indicates the verb is not in the AGTP-API approved method catalog; 460 indicates the path violates AGTP-API path grammar. The structural rejection model and the curated method catalog are specified in [AGTP-API].

Code	Name	Meaning
200	OK	Method executed successfully
202	Accepted	Method accepted; execution is asynchronous
204	No Content	Method executed; no response body
261	Negotiation In Progress	PROPOSE evaluation in progress; agent *MAY* poll for terminal status. See [AGTP-API].
262	Authorization Required	Request requires credential establishment, additional authorization scope, or consent that is not yet present. Covers PROPOSE-time authorization, missing scope at endpoint dispatch, ad-hoc method invocation requiring wildcards consent, and discovery requests blocked by anonymous-discovery policy. See [AGTP-API].
263	Proposal Approved	PROPOSE accepted; the proposed endpoint has been synthesized and instantiated. Response body carries the synthesized endpoint contract. See [AGTP-API].
400	Bad Request	Malformed AGTP request
401	Unauthorized	Authentication failure. Covers Agent-ID not recognized or not authenticated, INSPECT read-access ACL failure for unauthenticated callers, lifecycle-method invocation without a verified client certificate when genesis_issuer mode is in effect, and external-credential failures (e.g., missing Authorization header on a method that requires one, or an Authorization header that fails validation). The response body *SHOULD* carry a structured reason from the vocabulary agent-unauthenticated, oauth-required, oauth-invalid, acl-anonymous-blocked, or genesis-issuer-cert-required to disambiguate the failure mode. See Section 8.6 for the OAuth-specific

		reasons.
403	Forbidden	Agent lacks authority for requested action per Authority-Scope
404	Not Found	The path does not exist on this server. No endpoints are registered under it.
405	Method Not Allowed	The method is recognized and the path is valid, but the server's policy or registry does not expose this combination. The response body *MUST* list allowed methods for the path and any redirects from the manifest's policies.methods sub-block. The agent *MAY* PROPOSE the combination if it is not exposed by policy. AGTP-specific. See [AGTP-API].
408	Timeout	TTL exceeded before method could execute. AGTP-specific semantics; see below.
409	Conflict	Method conflicts with current state
410	Gone	Agent has been REVOKEd; canonical Agent-ID is permanently retired (status retired). A Deprecated agent (status deprecated) does *NOT* return 410 and continues to serve traffic; deprecation signals planned end-of-life rather than immediate retirement. AGTP-specific semantics; see below.
422	Unprocessable	Request well-formed but semantically invalid
429	Rate Limited	Agent is exceeding permitted request frequency
455	Scope Violation	Requested action is outside declared scope. Generic scope-violation code for cases not covered by the dedicated scope codes (429 rate-limit, 456 budget, 457 zone, 262 authority). Token-based or query-based scope violations are typical applications. AGTP-specific.
456	Budget	Method execution would exceed the Budget-

	Exceeded	Limit declared in the request. AGTP-specific.
457	Zone Violation	Request would route outside the AGTP-Zone-ID boundary. SEP-enforced. AGTP-specific.
458	Counterparty Unverified	PURCHASE counterparty failed merchant identity verification: Merchant-ID absent, Merchant-Manifest-Fingerprint mismatch, or merchant in non-Active lifecycle state. AGTP-specific. See [AGTP-MERCHANT].
459	Method Violation	The method name is not in the AGTP-API approved method catalog. The method itself is the problem. AGTP-specific. See [AGTP-API].
460	Endpoint Violation	The endpoint path violates AGTP-API path grammar. A path segment matches an approved method name, indicating method-name leakage into the path. AGTP-specific. See [AGTP-API].
461	RCNS Contract Available	The requested endpoint is not registered but the server is prepared to synthesize a contract for it. The response body carries a contract preview the caller may accept or decline. Returned in the confirm-first RCNS delivery mode. AGTP-specific. See [AGTP-API].
462	Reserved	Reserved for AGTP expansion.
463	Proposal Rejected	The service cannot or will not instantiate the proposed endpoint. Returned in response to PROPOSE. Response body *MUST* carry a structured reason (e.g., out-of-scope, policy-refused, composition-impossible, ambiguous, synthesis-disabled) and *MAY* carry an optional counter-proposal. AGTP-specific. See [AGTP-API].
464	RCNS No Contract	An RCNS synthesis attempt was made but no contract could be delivered. Response body *MUST* carry a structured reason:

		rcns-disabled (RCNS policy off on this server), trust-tier-insufficient (caller's resolved trust tier is below the server's RCNS minimum), composition-impossible (no synthesis path exists from registered primitives), synthesis-error (synthesis attempt failed at runtime), contract-not-yours (presented synthesis_id was issued to a different agent), or contract-revoked (presented synthesis_id was revoked). AGTP-specific. See [AGTP-API].
465	Reserved	Reserved for AGTP expansion.
500	Server Error	Internal failure in the responding system
503	Unavailable	Responding agent or system temporarily unavailable or Suspended
550	Delegation Failure	A delegated sub-agent failed to complete the requested action. AGTP-specific.
551	Authority Chain Broken	Delegation chain contains an unverifiable or broken identity link. AGTP-specific.
552	Reserved	Reserved for AGTP expansion.
553	Reserved	Reserved for AGTP expansion.
554	Reserved	Reserved for AGTP expansion.
555	Reserved	Reserved for AGTP expansion.

Table 6: AGTP Status Codes

The four structural rejection codes (404, 405, 459, 460) form the contract-level structural failure surface and have distinct recovery semantics. A 459 tells the agent to choose a different method from the AGTP-API catalog. A 460 tells the agent to restructure the path so that no path segment is a method name. A 405 tells the agent the method and path are each individually valid but the failure is a server policy decision; a different method-and-path combination might work, or PROPOSE might negotiate access. A 404 tells the agent the path does not exist on this server at all. Beyond the structural surface, 463 (Proposal Rejected) is the runtime negotiation rejection: the agent's PROPOSE was structurally valid but the server cannot or will not synthesize the requested endpoint.

Status code 262 (Authorization Required) consolidates the authority-related rejection conditions an agent might encounter when interacting with a server. It is returned when: (a) a PROPOSE request requires credential establishment before the server will evaluate it; (b) an endpoint invocation requires Authority-Scope that the agent has not declared; (c) an ad-hoc method invocation requires wildcards consent (wildcards: true on the agent identity document and wildcards_accepted: true in server policy) that is absent on either side; or (d) a discovery request is blocked by server policy that requires authenticated identity for manifest retrieval. The response body *MUST* identify which condition applies so the agent knows what to remediate.

Status code 455 (Scope Violation) is a generic scope-violation signal for cases not covered by the dedicated scope codes (429 rate-limit, 456 budget, 457 zone, 262 authority). Typical applications include token-based scope violations and query-based scope violations, where an operator defines a scope dimension outside the standard set. Authority-Scope violations specifically are signaled with 262, not 455. Status code 456 (Budget Exceeded) is the dedicated code for resource-consumption scope violations: the agent's requested action would consume resources beyond what the principal authorized for this invocation. Status code 457 (Zone Violation) is returned by SEPs when a DELEGATE request would route to an agent outside the declared zone boundary. Status code 458 (Counterparty Unverified) is returned on PURCHASE invocations when the receiving server cannot verify that the requesting agent has performed valid merchant identity verification against the server's current Merchant Manifest Document, or when the merchant is in a non-Active lifecycle state; see [AGTP-MERCHANT].

Status code 551 (Authority Chain Broken) is returned when a server processing a delegated request cannot reconstruct a valid delegation sequence from the delegating agent to the requesting agent. The specific unverifiable link *SHOULD* be identified in the response

body. Status code 408 (Timeout) is reused from HTTP with AGTP-specific semantics: it signals that the method's declared TTL expired before execution completed, distinct from HTTP's request-timeout semantics. Status code 410 (Gone) is reused from HTTP with AGTP-specific semantics: it indicates that an Agent-ID has been permanently retired through REVOKE of its Agent Genesis, distinct from HTTP's resource-removed semantics. A Deprecated agent does *NOT* return 410; deprecation signals planned end-of-life while the agent continues to serve traffic (see DEPRECATE in Section 7.2.18). The canonical Agent-ID of a retired agent *MUST NOT* be retried. All AGTP-specific status codes are operational signals, not protocol errors, and *MUST* be logged for audit purposes.

5.7. Wire Format and Content-Type

AGTP request and response bodies are encoded as JSON or YAML. The following media types are defined by AGTP. Their IANA registration status varies; see the IANA Considerations section for the authoritative status table.

Media Type	Use	IANA Status
application/ vnd.agtp+json	AGTP method request/ response bodies (JSON)	Planned (this document)
application/ vnd.agtp+yaml	AGTP method request/ response bodies (YAML)	Planned (this document)
application/ vnd.agtp.identity+json	Agent Identity Document (JSON)	Vendor-tree registration submitted
application/ vnd.agtp.identity+yaml	Agent Identity Document (YAML)	Vendor-tree registration submitted

Table 7: AGTP Media Types Defined in This Document

Two additional media types are defined in the companion AGTP-API specification [AGTP-API] and registered concurrently: application/ vnd.agtp.manifest+json (server manifest) and application/ vnd.agtp.endpoint+json (endpoint definition). Both are listed in the master Media Type Registry section of this document for cross-document discoverability.

Implementations *MUST* include the appropriate Content-Type on all AGTP requests and responses that carry a message body. Responses with no body (e.g., 204 No Content) *MUST NOT* include a Content-Type header. Binary or streaming extensions *MAY* define additional Content-Type values as part of their companion specifications.

EXECUTE method invocations carry application-layer payloads whose Content-Type is set by the carried application protocol (for example, application/vnd.mcp.tools+json for MCP tool invocations carried over AGTP). The AGTP server dispatches based on the carried Content-Type; see Section 7.2.8 and Section 8.

The common structure for AGTP method-level request bodies:

```
{
  "method": "QUERY",
  "task_id": "task-0042",
  "session_id": "sess-a1b2c3d4",
  "parameters": { },
  "context": { }
}
```

And for AGTP method-level response bodies:

```
{
  "status": 200,
  "task_id": "task-0042",
  "result": { },
  "attribution": { }
}
```

5.8. Early Implementations

AGTP is a proposed specification. No production implementations exist at the time of this writing. The author encourages early prototype implementations to validate the protocol design, identify gaps, and generate feedback prior to IETF working group submission.

> If you are building an AGTP prototype or reference implementation, please share your findings via the feedback channel listed on the cover of this document. A reference implementation in Python and/or Go is planned as open-source software concurrent with or shortly after IETF I-D submission. Implementation reports are welcome and will be incorporated into subsequent draft revisions.

Implementers wishing to experiment before final IANA port assignment *SHOULD* use a locally-chosen port from the Dynamic Ports range (49152-65535) on developer-controlled endpoints. Implementations *MUST NOT* publish or document any specific port number as an AGTP-associated value until IANA assignment is complete.

ACTIVATE is one of the three Lifecycle methods on the eighteen-method floor; see Section 7.2.14. Earlier revisions of this document described ACTIVATE as an optional method extension carried in a separate companion. With the promotion of the Lifecycle group to the embedded floor, ACTIVATE, DEACTIVATE, and REVOKE are all core methods; any conformant AGTP implementation *MUST* support them. Package-binding semantics specific to .nomo governed packages remain governance-platform-specific and *MAY* be layered on top of the protocol-level ACTIVATE transaction without affecting the wire contract.

6. Agent Identity, URI Structure, and Registration

6.1. URI Structure and Resolution Mechanics

6.1.1. Foundational Principle

AGTP identity is agent-first. Every agent is identified by a canonical Agent-ID: a 256-bit cryptographic identifier derived from the agent’s Agent Genesis hash at ACTIVATE time. The canonical Agent-ID is the authoritative identifier in every AGTP protocol operation. It appears in the Agent-ID header of every request, is the key in the registry, and is the cross-layer reference linking the AGTP Agent Certificate extension to the governance-layer Agent Genesis.

All other agent identification forms recognized by AGTP, including domain-anchored URIs and Web3 resolution targets, are aliases that resolve to a canonical Agent-ID. In the event of any conflict between an alias and a canonical Agent-ID, the canonical Agent-ID *MUST* be treated as authoritative.

AGTP URIs are addresses, not filenames. File format suffixes (.agtp) *MUST NOT* appear in canonical agtp:// URIs. A URI resolves to an Agent Identity Document or Agent Namespace Document derived from the underlying package; it does not expose or serve the package itself.

Implementations *MUST* treat any URI containing a file extension in the path as non-canonical and *SHOULD* issue a 301 Moved Permanently redirect to the canonical form prior to resolution.

The .agent suffix and the .nomo suffix are file format designators for AGTP agent packages; they are not URI hostname labels. Earlier drafts admitted hostname forms ending in .agent or .nomo as agent-native hierarchical TLDs; this revision removes that admission. Hostnames in AGTP URIs are standard DNS hostnames per RFC 3986.

6.1.2. Canonical URI Forms

AGTP is identity-first. Agent-IDs are canonical and content-addressed; hosting is incidental. Form 1 (agtp://[Agent-ID]) is the canonical identity form. Form 1a (agtp://[Agent-ID]@[host]) provides direct addressing for cases where the resolver does not yet know how to reach the canonical ID, and bridges the gap between identity-first addressing and the existing DNS-based reachability infrastructure.

Forms 2 and 2a address servers and organizations rather than specific agents, providing entry points for discovery operations that do not target a named agent. Forms 3 and 4 address agents by local name within a domain's namespace; they differ only in deployment convention.

Form 1. Canonical identity (cryptographic, authoritative):
agtp://[Agent-ID]

Form 1a. Canonical identity with explicit host (direct addressing):
agtp://[Agent-ID]@[host]

Form 2. Server-level discovery (specific server endpoint):
agtp://[host]

Form 2a. Organization-level discovery (DNS-registered domain root):
agtp://[domain]

Form 3. Domain-anchored agent (domain owns the namespace):
agtp://[domain]/agents/[agent-name]

Form 4. Subdomain-anchored agent (dedicated AGTP subdomain):
agtp://agtp.[domain]/agents/[agent-name]

The port portion of any AGTP URI is *OPTIONAL*. When omitted, AGTP clients *MUST* use the IANA-assigned default port 4480. The port is included in URIs only for non-default deployments and appears in the host component (Form 1a host or Form 2 host); ports *MUST NOT* appear in Form 2a domain components or in Forms 3/4 domain components.

6.1.2.1. Form 1 — Canonical Identity

Form 1 carries only the 64-character lowercase hex representation of the Agent Genesis SHA-256 hash:

agtp://7f3a9c2d...e1f8b0a4

Form 1 resolves to a signed Agent Identity Document through any verification path declared in the agent's registry record (Section 5.2). The canonical ID is self-describing: any AGTP-aware governance platform, transparency log, or resolution service can return the Identity Document given the canonical ID alone, without prior knowledge of which organization, domain, or blockchain the agent is registered under.

6.1.2.2. Form 1a — Canonical Identity with Explicit Host

Form 1a embeds an explicit host alongside the canonical Agent-ID:

agtp://7f3a9c2d...e1f8b0a4@agents.acme.com
agtp://7f3a9c2d...e1f8b0a4@192.0.2.42
agtp://7f3a9c2d...e1f8b0a4@agents.acme.com:9999

The Agent-ID identifies; the host tells the resolver where to reach a server that can return the Identity Document or accept method invocations. Form 1a is the preferred form when:

* The agent has been issued a canonical Agent-ID but is not yet registered with an AGTP discovery service.

* The resolver does not yet have a path from canonical Agent-ID to endpoint (no DNS anchor, no transparency log entry, no governance-platform resolution).

* A client wishes to address an agent by ID directly without round-tripping through a registry.

Form 1a does not weaken the identity-first model. The canonical Agent-ID remains authoritative; the host portion is treated as a resolution hint and *MUST NOT* be used to derive identity. If the host returns an Identity Document whose agent_id field does not match the canonical Agent-ID in the URI, the response *MUST* be rejected.

6.1.2.3. Form 2 — Server-Level Discovery

Form 2 addresses a specific server endpoint without naming an agent:

agtp://agents.acme.com
agtp://agtp-edge-7.us-east.acme.com
agtp://192.0.2.42
agtp://[2001:db8::42]
agtp://agents.acme.com:9999

The host component *MAY* be any RFC 3986 host: a registered hostname, an FQDN, an IPv4 address, an IPv6 address, optionally followed by a port. Form 2 is used for diagnostic operations addressing a specific server instance (a particular edge node, a specific deployment in a load-balanced fleet, a server identified by IP) and for method invocations that target server-level state rather than agent-level state.

6.1.2.4. Form 2a — Organization-Level Discovery

Form 2a addresses an organization's AGTP presence via its registered DNS domain:

agtp://acme.com
agtp://example.org

The domain component *MUST* be a DNS-registered domain name with at least one label. IP addresses, ports, and userinfo components *MUST NOT* appear in Form 2a; URIs carrying any of these are Form 2, not Form 2a.

Form 2a resolution proceeds via standard DNS lookup of the domain, then AGTP connection establishment on port 4480 against the resolved address. The response is the organization's canonical server manifest. Whether the domain resolves to a single AGTP server, a load-balanced fleet, or a CDN-fronted edge is a deployment concern; Form 2a does not commit the caller to any specific topology.

The syntactic distinction between Form 2 (any RFC 3986 host, possibly with port) and Form 2a (bare DNS domain) corresponds to a semantic distinction. Form 2 addresses a specific reachable server; Form 2a addresses whichever server the organization's DNS currently points at. Both are valid; they differ in caller intent.

6.1.2.5. Forms 3 and 4 — Domain-Anchored Agents

Forms 3 and 4 address an agent by local name within a domain's agent namespace:

Form 3: agtp://acme.com/agents/bookbot
Form 4: agtp://agtp.acme.com/agents/bookbot

Form 3 places the agent under the organization's primary domain. Form 4 places the agent under a dedicated agtp. subdomain. Resolution semantics are identical: the AGTP server at the domain (or subdomain) consults its hosted_agents manifest entries and returns the canonical Agent-ID for the named local agent. The local agent name is unique within the domain's namespace; the same name under a different domain is a different agent.

Forms 3 and 4 differ only in deployment convention. Operators choose based on their infrastructure preferences: organizations that prefer to keep agent-protocol traffic on a dedicated subdomain use Form 4; organizations that operate AGTP as the canonical face of their primary domain use Form 3. The protocol treats them as equivalent.

In both forms the domain component *MUST* be a DNS-registered domain name; IPs and ports *MUST NOT* appear. Direct addressing with explicit host is available via Form 1a if the resolver knows the canonical Agent-ID.

6.1.3. URI Grammar

The URI grammar is defined in ABNF as:

```
AGTP-URI     = "agtp://" agtp-locator
agtp-locator = agent-id [ "@" host ]                ; Form 1, 1a
             / host                                   ; Form 2
             / domain                                 ; Form 2a
             / domain "/agents/" agent-name           ; Form 3
             / "agtp." domain "/agents/" agent-name   ; Form 4
agent-id     = 64HEXDIG                              ; lowercase hex
host         = IP-literal / IPv4address / reg-name [ ":" port ]
                                                     ; per RFC 3986
domain       = label *("." label)                    ; DNS-registered, no port
label        = ALPHA *( ALPHA / DIGIT / "-" )
agent-name   = 1*( ALPHA / DIGIT / "-" / "_" )
port         = 1*DIGIT
                                                     ; OPTIONAL; defaults to 4480
```

AGTP URIs are addresses, not filenames. File extensions in the path *MUST NOT* appear in canonical agtp:// URIs. A URI resolves to an Agent Identity Document, a server manifest, or an endpoint response derived from server state; it does not expose or serve a package file.

Implementations *MUST* treat any URI containing a file extension in the path as non-canonical and *SHOULD* issue a 301 Moved Permanently redirect to the canonical form prior to resolution.

6.1.4. Method-on-URI Invocation Pattern

AGTP method invocations target a URI plus an optional endpoint path. The conceptual pattern is:

METHOD agtp://[locator]/[endpoint-path]

The locator addresses an agent (Forms 1, 1a, 3, 4), a server (Form 2), or an organization (Form 2a). The endpoint path is the path portion of the request line as specified in Section 5.7. For server-level and organization-level discovery operations the endpoint path is omitted (target-less DISCOVER); for agent-level operations the endpoint path identifies the endpoint to invoke.

The wire-level encoding of this pattern is the AGTP request line (method and path) plus the connection target (host derived from the URI). The URI is the agent-facing addressing notation; the wire format is what travels over TLS.

6.1.5. Web3 Verification Anchors

AGTP supports Web3-anchored verification paths for canonical Agent-IDs whose underlying Agent Genesis is registered through a blockchain-based verification service. The verification path is declared in the agent’s registry record per Section 6.1.9 and does not affect URI syntax: a Web3-anchored agent is addressed by its canonical Agent-ID via Form 1 or Form 1a like any other agent.

Integration with specific Web3 naming and resolution systems is specified in [AGTP-WEB3].

6.1.6. Non-Canonical Forms and Redirect Behavior

The following non-canonical forms *SHOULD* be redirected to their canonical equivalents. Implementations *MUST NOT* serve package contents in response to any URI form.

Received URI	Canonical Redirect Target
agtp://acme.tld/agents/ customer-service.agent	agtp://acme.tld/agents/ customer-service
agtp://acme.tld/agents/ customer-service.nomo	agtp://acme.tld/agents/ customer-service
agtp://acme.tld/agents/ customer-service.agtp	agtp://acme.tld/agents/ customer-service

Table 8: Non-Canonical URI Forms and Redirect Targets

6.1.7. Query Parameters for Format Selection

All AGTP URI resolution requests accept an optional format query parameter controlling the serialization of the returned document.

Query Parameter	Returned Representation
(none)	Agent Identity Document, human-readable application/vnd.agtp+json
?format=manifest	Agent Identity Document, human-readable application/vnd.agtp+json
?format=json	Agent Identity Document, compact application/vnd.agtp+json
?format=certificate	Agent Genesis fields only, application/vnd.agtp+json
?format=status	Lifecycle state and operational status only, application/vnd.agtp+json

Table 9: AGTP URI Format Query Parameters

All format variants return signed application/vnd.agtp+json content. The ?format=json parameter is intended for programmatic consumers. The default returns the full human-readable manifest suitable for browser rendering by an AGTP-aware client.

6.1.8. Resolution Mechanics

AGTP URI resolution proceeds according to the URI form presented. Form 1 and Form 1a (canonical agent identity) resolve through a governance-platform registry or transparency-log lookup. Forms 2 and 2a (server and organization discovery) resolve through direct AGTP connection establishment to the host or domain. Forms 3 and 4 (domain-anchored agents) resolve through DNS to an AGTP server whose hosted_agents manifest maps the URI's local agent name to a canonical Agent-ID.

6.1.8.1. Form 1 Resolution (Canonical ID)

When an AGTP resolver receives a URI of the form agtp://[256-bit-hex-id], it *MUST* perform the following steps:

1. Parse and validate the canonical Agent-ID. If the identifier is malformed (length, character set), return 400 Bad Request with error code invalid-canonical-id.

2. Query the agent's governance platform registry for the record associated with the canonical Agent-ID. If the resolver does not know which governance platform holds the record, it *MAY* query a transparency log per Section 5.2 to locate the record.

3. Verify the registry record lifecycle state. If suspended, return 503 Service Unavailable with lifecycle state in the response body. If retired, return 410 Gone with lifecycle state and revocation timestamp. If deprecated, the resolver *SHOULD* continue with resolution; the registry record *MUST* carry the deprecation signal so the relying party can surface it to clients and act on any successor_agent_id declared in the deprecation event.

4. Retrieve the agent's package (.agent or .nomo) from the package store referenced by the registry record.

5. *Verify the package integrity hash before proceeding.* If integrity verification fails, return 500 Internal Error with error code package-integrity-failure. *MUST* be logged.

6. Extract the embedded manifest from the verified package.

7. Sign the Identity Document using the governance platform's signing key. Return the signed application/vnd.agtp+json document in the format specified by the query parameter.

Form 1 resolution does not require prior knowledge of an organization domain, a DNS record, or a Web3 naming anchor. The canonical Agent-ID is sufficient input.

6.1.8.2. Forms 2 and 2a Resolution (Server and Organization Discovery)

When an AGTP resolver receives a URI of Form 2 (agtp://[host]) or Form 2a (agtp://[domain]), it *MUST* perform the following steps:

1. Parse and validate the URI. Form 2 admits any RFC 3986 host with optional port. Form 2a admits a DNS-registered domain name with at least one label and *MUST NOT* include port or userinfo components.

2. Open an AGTP/TLS connection to the host or domain. For Form 2a, resolve the domain via DNS first. For both forms, AGTP traffic targets port 4480 unless an explicit non-default port appears in the URI (Form 2 only).

3. Issue the agent’s intended method (typically DISCOVER for discovery operations) with no agent-identifying target.

4. The server returns its server manifest (for Form 2 or Form 2a addressed against a server’s primary endpoint) or whatever server-level response the invoked method produces.

Forms 2 and 2a do not resolve to a canonical Agent-ID; they target server or organization state directly. They do not participate in the canonical-Agent-ID identity model.

6.1.8.3. Forms 3 and 4 Resolution (Domain-Anchored Agents)

When an AGTP resolver receives a URI of Form 3 or Form 4, it *MUST* perform the following steps:

1. Parse and validate the URI. If the URI is malformed, return 400 Bad Request with error code invalid-uri-form.

2. Resolve the domain (Form 3) or agtp.[domain] subdomain (Form 4) via DNS, then open an AGTP/TLS connection to port 4480.

3. Look up the agent-name from the URI path against the server’s hosted_agents manifest entries to obtain the canonical Agent-ID. If no matching agent is found, return 404 Not Found.

4. Continue with Form 1 resolution steps 3 through 7 using the resolved canonical Agent-ID.

The package's executable content, code, logic, and any fields not included in the Identity Document schema *MUST NOT* be returned at any step of any resolution path. URI resolution exposes identity and status exclusively.

6.1.9. Trust Tiers and Verification Paths

The Agent Genesis carries a trust_tier field (one of 1 Verified, 2 Org-Asserted, 3 Experimental) and a verification_path field (one of dns-anchored, log-anchored, hybrid, or org-asserted) recording how the agent's identity was anchored at ACTIVATE time. These fields are surfaced in the Agent Identity Document and consulted by Scope-Enforcement Points, governance gateways, and peer agents during runtime authority decisions.

Tier 1 agents are eligible for the full Authority-Scope vocabulary, delegation chains, financial transactions, and multi-organization collaboration. Tier 1 verification requires exactly one of three verification paths (dns-anchored, log-anchored, hybrid) to succeed at ACTIVATE time, each backed by distinct evidence (DNS TXT record, transparency log inclusion proof, or DNS + blockchain signature combination). All Tier 1 paths produce identity attestations of equivalent strength.

Tier 2 agents declare organizational affiliation without cryptographic verification (verification_path: org-asserted) and carry the trust_warning: "verification-incomplete" signal. Tier 3 agents are experimental and confined to development environments.

The normative verification procedures, evidence requirements, tier assignment rules, and Authority-Scope eligibility constraints are specified in [AGTP-TRUST].

6.1.10. Subdomain Deployment Pattern

Organizations *SHOULD* deploy AGTP endpoints at a dedicated subdomain following the pattern agtp.[organization-domain.tld] (e.g., agtp.acme.tld). This is the recommended enterprise deployment pattern: it provides clean separation between web and agent infrastructure, allows independent certificate management for the AGTP endpoint, and is consistent with service-specific subdomain conventions. An organization with an AGTP subdomain *SHOULD* also configure their primary domain to redirect AGTP requests:

```
agtp://acme.tld/agents/customer-service
  → 301 → agtp://agtp.acme.tld/agents/customer-service
```

6.1.11. The /agents/ Reserved Path Prefix

The path prefix /agents/ is reserved in all agtp:// URIs for agent namespace operations. Implementations *MUST* support this prefix. The registry root at /agents (no trailing label) resolves to the Agent Namespace Document (see Section 5.4).

6.1.12. Collision Prevention

The canonical Agent-ID is the collision-prevention primitive. Two canonical Agent-IDs are distinct if and only if the 256-bit identifiers differ, and the governance platform enforces uniqueness at issuance time by deriving the ID from the Agent Genesis hash.

For alias forms, collision prevention operates at the namespace level. agtp://acme.com/agents/customer-service and agtp://chrishood.com/agents/customer-service resolve to distinct canonical Agent-IDs because they are registered under different DNS domains. Within a single governance zone, the governance platform enforces uniqueness of agent local names at registration time.

Infrastructure *MUST* use the canonical Agent-ID for all routing, logging, and attribution operations. Alias URIs are a display and discovery layer only. An alias that resolves to a canonical Agent-ID different from the one carried in the Agent-ID header on a request *MUST* cause the request to be rejected with 401 Unauthorized and *MUST* be logged.

6.1.13. IANA Considerations for the agtp:// URI Scheme

The agtp:// URI scheme is registered with IANA per [RFC7595]. Registration template:

URI scheme name: agtp

Status: Permanent

URI scheme syntax: agtp://[Agent-ID] (canonical identity, Form 1) agtp://[Agent-ID]@[host] (canonical identity with explicit host, Form 1a) agtp://[host] (server-level discovery, Form 2) agtp://[domain] (organization-level discovery, Form 2a) agtp://[domain]/agents/[agent-name] (domain-anchored agent, Form 3) agtp://agtp.[domain]/agents/[agent-name] (subdomain-anchored agent, Form 4)

URI scheme semantics: Identifies an AI agent, an AGTP server, or an

organizational AGTP presence operating over the Agent Transfer Protocol. The authoritative agent identification form (Form 1) uses a 256-bit hex-encoded cryptographic identifier derived from the agent's Agent Genesis. Form 1a augments the canonical ID with an explicit host to enable direct addressing in deployments where the canonical ID has not yet been resolved through a registry. Forms 2 and 2a address servers and organizations respectively for discovery operations that do not target a specific agent. Forms 3 and 4 address agents by local name within a domain's namespace and resolve to canonical Agent-IDs through the server's hosted_agents manifest. Forms 3 and 4 differ only in deployment convention.

Applications/protocols that use this URI scheme: Agent Transfer Protocol (this document)

Interoperability considerations: The canonical Agent-ID form is the authoritative agent identity representation. Form 1a embeds a resolution hint alongside the canonical Agent-ID and *MUST NOT* be used to derive identity. Forms 3 and 4 resolve through DNS to an AGTP server endpoint whose hosted_agents manifest maps the URI path component to a canonical Agent-ID. Forms 2 and 2a do not resolve to canonical Agent-IDs; they target server or organization state directly. Implementations *MUST* accept canonical Agent-IDs (Form 1) and *SHOULD* support at least one of the domain-anchored agent forms (Form 3 or Form 4). The port portion of any AGTP URI is OPTIONAL and defaults to the IANA-assigned port 4480 when omitted; ports *MAY* appear in Form 1a and Form 2 only. File format suffixes (.agtp, .agent, .nomo) *MUST NOT* appear in agtp:// URIs.

Contact: Chris Hood, chris@nomotic.ai

References: This document

The agtp:// URI scheme registration is open and unencumbered. No intellectual property claims apply to the URI scheme itself.

6.1.14. .well-known Bootstrap

Organizations operating an AGTP namespace under a DNS domain *SHOULD* publish a bootstrap document at the well-known URI https://{domain}/.well-known/agtp per [RFC8615]. The document is returned over HTTPS and declares the organization's AGTP presence, the endpoint at which AGTP traffic should be directed, and any verification anchors the organization wishes to surface for discovery.

The bootstrap document is JSON with the following fields:

```
{
  "agtp_version": "0.7",
  "endpoint": "agtp://agtp.acme.tld/agents",
  "namespace_root": "agtp://acme.tld/agents",
  "verification": {
    "dns_anchor": "_agtp.acme.tld",
    "log_anchor": null
  },
  "issuer": "https://ca.acme.tld",
  "discovery": {
    "namespace_document": "https://agtp.acme.tld/agents.json",
    "well_known_updated_at": "2026-04-30T00:00:00Z"
  }
}
```

Resolvers encountering an agtp:// URI for a domain they have not previously interacted with *SHOULD* retrieve the bootstrap document to learn the canonical AGTP endpoint and verification anchors before attempting AGTP traffic. The bootstrap is advisory; absence of a .well-known/agtp document does not indicate that the domain has no AGTP presence, only that the domain has not chosen to advertise one through this mechanism.

A .well-known/agtp document returning HTTP 404 is a positive declaration that the domain operates no AGTP namespace. Resolvers *MAY* cache this negative result for the period indicated by the HTTP response's Cache-Control directive, with a default of 24 hours when no directive is present.

6.2. Agent Namespace Document

6.2.1. Purpose and Scope

The Agent Namespace Document is the index of all Active agents registered under an organization's governance zone. It is returned in response to a request targeting the /agents path:

agtp://acme.tld/agents
agtp://agtp.acme.tld/agents

The Agent Namespace Document is not a manually editable file. It is generated and cryptographically signed by the governance platform each time the registry changes. Any Namespace Document that fails signature verification *MUST* be rejected by the requesting party.

6.2.2. Document Schema

```
{
  "document_type": "agtp-namespace",
  "schema_version": "1.0",
  "org_domain": "acme.tld",
  "governance_zone": "zone:acme-internal",
  "generated_at": "2026-03-20T14:00:00Z",
  "signature": {
    "algorithm": "ES256",
    "key_id": "agtp-gov-key-acme-01",
    "value": "[base64-encoded-signature]"
  },
  "agents": [
    {
      "agent_label": "customer-service",
      "canonical_id": "3a9f2c1d8b7e4a6f...",
      "lifecycle_state": "Active",
      "trust_tier": 1,
      "cert_status": "Active",
      "manifest_uri": "agtp://agtp.acme.tld/agents/customer-service",
      "activated_at": "2026-01-15T09:00:00Z",
      "last_updated": "2026-03-01T11:30:00Z"
    }
  ],
  "total_active": 1,
  "namespace_cert_fingerprint": "b2c4d6e8..."
}
```

Figure 3: Agent Namespace Document Schema

The agents array *MUST* include only agents in Active lifecycle state. Suspended, Revoked, and Deprecated agents *MUST NOT* appear in the Namespace Document.

6.2.3. Integrity and Freshness

The Namespace Document *MUST* include a generated_at timestamp. Implementations *SHOULD* treat Namespace Documents older than a configurable freshness threshold (default: 300 seconds) as stale and re-request. The governance platform *MUST* re-sign the Namespace Document within 60 seconds of any registry change.

The signature covers the entire document including generated_at. Replaying an older signed Namespace Document to conceal a revocation event is a known attack vector; implementations *MUST* reject Namespace Documents with a generated_at timestamp older than the freshness threshold.

6.3. Agent Identity Document and the .agtp Format

6.3.1. Purpose and Scope

The Agent Identity Document is the protocol's canonical representation of a specific agent's identity, status, and behavioral scope. Prior versions of this specification referred to this artifact as the Agent Manifest Document; v07 renames it to the Agent Identity Document to reflect its IANA-registered media type (application/vnd.agtp.identity+json) and to clarify its role as the identity primitive of the protocol. The Identity Document is returned in response to any AGTP URI resolution request targeting a specific agent:

```
agtp://[canonical-agent-id]
agtp://acme.tld/agents/customer-service
agtp://acme.tld/agents/customer-service?format=json
```

The Identity Document is derived from the embedded manifest inside the agent's .agent or .nomo package. It is not a separate file that can be independently modified. The governance platform *MUST* verify the package integrity hash before extracting and serving the Identity Document.

6.3.2. The Three Document Formats and Their Relationship

```
.agent   ———— open packaging format (patent pending, Chris Hood)
   |           Contains: manifest + integrity hash + behavioral
   |           trust score + executable content
   |
.nomo    ———— governed packaging format (patent pending, Chris Hood)
   |           Contains: .agent contents + CA-signed cert chain
   |           + governance zone binding
   |
  Both produce ↓ when queried via agtp:// URI
   |
.agtp    ———— wire-level Identity Document (this spec)
             Contains: identity fields, lifecycle state, trust
             tier, trust score, behavioral scope, methods,
             capabilities, verification anchors
             Open format defined by AGTP specification
             Content-Type: application/vnd.agtp.identity+json
```

Figure 4: Relationship Between AGTP File Formats

The .agtp format is the protocol's own document type. It is what gets transmitted over the wire. The .agent and .nomo formats are what get deployed. An AGTP implementation is not required to understand .agent or .nomo packaging internals; it is only required to produce and consume .agtp Identity Documents.

Additional packaging formats *MAY* be defined by third parties, provided they can produce conformant .agtp Identity Documents when queried. Such formats *MUST* implement the integrity verification requirement: the Identity Document *MUST* be derived from a verified package, not from an independently stored or editable file.

6.3.3. Agent Identity Document Schema

The Agent Identity Document carries three categories of fields: *REQUIRED* fields that every Identity Document *MUST* contain; *RECOMMENDED* fields that *SHOULD* be present when the corresponding information is available; and *CONDITIONAL* fields that *MUST* be present when the conditions described later in this section are met.

The following fields are *REQUIRED* in all Agent Identity Documents:

```
{
  "agtp_version": "0.7",
  "document_type": "agtp-identity",
  "document_version": "1.0",
  "agent_id": "3a9f2c1d8b7e4a6f0c2d5e9b1a3f7c0d...",
  "name": "customer-service",
  "description": "Handles customer service requests.",
  "principal": "Acme Corporation",
  "principal_id": "acme.tld",
  "issuer": "https://ca.acme.tld",
  "issued_at": "2026-01-15T09:00:00Z",
  "updated_at": "2026-03-01T11:30:00Z",
  "status": "Active",
  "methods": [
    "QUERY", "DESCRIBE", "SUMMARIZE",
    "EXECUTE", "ESCALATE", "CONFIRM", "NOTIFY"
  ],
  "capabilities": [
    "customer-service:tier-1",
    "knowledge-base:read",
    "ticket-system:create"
  ],
  "scopes_accepted": [
    "documents:query",
    "knowledge:query",
    "calendar:book",
    "escalation:route"
  ],
  "trust_score": 0.94
}
```


Figure 5: Agent Identity Document - Required Fields

Field semantics:

agtp_version: The version of the AGTP protocol the agent speaks. SemVer string.

document_type: Self-identification of the document. *MUST* be agtp-identity for documents conforming to this specification.

document_version: The schema version of the Identity Document itself. SemVer string. Distinct from agtp_version. Permits the document schema to evolve independently of the protocol.

agent_id: The 256-bit canonical Agent-ID, hex-encoded, derived from the agent’s Agent Genesis hash. Authoritative in every AGTP protocol operation.

name: A human-readable label for the agent. Not unique across the AGTP ecosystem; uniqueness is provided by agent_id.

description: Short prose describing what the agent does.

principal: The human-readable name of the organization or individual that governs the agent.

principal_id: A stable, machine-readable identifier for the principal (typically the principal's primary DNS domain, or a canonical organization identifier).

issuer: The URL of the issuer that signed the Identity Document. The issuer's public key is published at a well-known location under this URL; signature verification details are specified in [AGTP-CERT].

issued_at: ISO 8601 timestamp at which the Identity Document was first issued (typically the moment of ACTIVATE).

updated_at: ISO 8601 timestamp at which the Identity Document was last updated. Promotes freshness checks. *MUST* be greater than or equal to issued_at.

status: The lifecycle state of the agent. One of: active, suspended, retired, deprecated. The state is mutated by the Lifecycle methods on the protocol floor (Section 7.1): ACTIVATE sets active, DEACTIVATE sets suspended, REINSTATE returns a suspended agent to active, REVOKE sets retired (permanent), DEPRECATE sets deprecated. Semantics:

* active — accepts AGTP traffic normally.

* suspended — refuses AGTP traffic with 503 Service Unavailable; recoverable via REINSTATE.

* retired — refuses AGTP traffic with 410 Gone; permanent; canonical Agent-ID is never reissued.

* deprecated — continues to accept AGTP traffic but signals planned end-of-life; clients *SHOULD* migrate.

methods: The set of AGTP methods the agent's server speaks. *MUST* include every method the server is prepared to accept. The protocol-defined eighteen-method floor (see Section 7.1) *MUST* be represented if the server claims AGTP conformance.

capabilities: Higher-level service descriptors of what the agent can

do. Distinct from methods: methods are the protocol verbs the server speaks; capabilities are the application-level competencies the agent exposes through those methods.

scopes_accepted: The Authority-Scope tokens the agent will accept on inbound requests. Format and semantics defined in Appendix C.

trust_score: A scalar value on the closed interval [0.0, 1.0] expressing behavioral trust assessed by the agent's principal or governance platform. Computation methodology, freshness requirements, and signature binding are specified in [AGTP-TRUST]. The base AGTP specification carries the field; AGTP-TRUST specifies how the value is produced and verified.

The following fields are *RECOMMENDED*:

```
{
  "trust_tier": 1,
  "verification_path": "dns-anchored",
  "owner_id": "nomotic.inc",
  "role": "agent",
  "org_domain": "acme.tld",
  "governance_zone": "zone:acme-internal",
  "cert_fingerprint": "b2c4d6e8...",
  "cert_status": "Active",
  "dns_anchor_record": "_agtp.acme.tld TXT agtp-zone=...",
  "log_inclusion_proof": null,
  "package_format": "nomo",
  "package_integrity_hash": "sha256:[hash]",
  "audit_log_uri": "agtp://agtp.acme.tld/audit/customer-service",
  "escalation_policy": "route-to-human-on-scope-limit",
  "delegation_permitted": false,
  "max_delegation_depth": 0,
  "manifest_issuer": "registrar.acme.tld",
  "manifest_issuer_public_key": "<base64url-encoded 32 bytes>",
  "manifest_signature": "<base64url-encoded Ed25519 signature>"
}
```

Figure 6: Agent Identity Document - Recommended Fields

trust_tier: Tier classification per [AGTP-TRUST]. One of 1, 2, or 3. Resolved at server load time per the precedence rule in [AGTP-TRUST]: explicit declaration in the Identity Document beats Genesis-derived fallback, which beats a conservative Tier 2 default.

verification_path: Verification path used at ACTIVATE time. One of

dns-anchored, log-anchored, hybrid, or org-asserted. Resolved per the precedence rule in [AGTP-TRUST] alongside trust_tier. The first three values are Tier 1 paths; org-asserted is the Tier 2 value. *MUST* match the verification_path field of the underlying Agent Genesis when one is loaded. Detailed semantics specified in [AGTP-CERT].

owner_id: Identifier of the human or organizational principal accountable for this agent. Resolved per the precedence rule in [AGTP-TRUST]: explicit declaration beats Genesis-derived fallback. Omitted when no Agent Genesis is loaded and the operator did not set an explicit value. Semantics specified in [AGTP-IDENTIFIERS]; stamped on responses as the Owner-ID response header per Section 5.5.4.

role: Capability role declared for this agent. One of agent (default) or merchant. The merchant value signals that the agent is a transactional counterparty addressable by PURCHASE and verified per [AGTP-MERCHANT]; agent is the default capability posture for all other agents. role is a capability attribute carried on the Agent Identity Document, not on the Agent Genesis: capability may change over an agent’s lifetime without changing identity. Operators *MAY* add or remove the merchant role without re-issuing the Agent Genesis or rotating the canonical Agent-ID. Future revisions *MAY* define additional role values. Implementations encountering an unknown role value *MUST* treat the agent as role: agent (the default) and log the unknown value for operator review.

org_domain: The DNS domain under which the agent is registered, when applicable.

governance_zone: The governance zone identifier for the agent’s deployment context.

cert_fingerprint, cert_status: AGTP Agent Certificate metadata. Specified in [AGTP-CERT].

dns_anchor_record: Populated when verification_path is dns-anchored or hybrid.

log_inclusion_proof: Populated when verification_path is log-anchored. Carries the transparency log inclusion proof per [RFC9162] or the COSE_Sign1 receipt per [RFC9943]. Detailed semantics specified in [AGTP-LOG].

package_format, package_integrity_hash: Deployment-layer metadata

identifying the package format and the hash that the Identity Document was extracted from.

audit_log_uri: AGTP URI of the agent's audit log resource.

escalation_policy, delegation_permitted, max_delegation_depth: Policy fields governing the agent's escalation and delegation behavior.

The following fields are *CONDITIONAL* and *MUST* be present when the conditions described below are met:

trust_warning: *REQUIRED* when trust_tier is 2, or when trust_score is below the threshold configured by the issuer. Value is a short string identifying the warning class (e.g., verification-incomplete, trust-score-degraded).

trust_explanation: *REQUIRED* when trust_warning is present. Value is a prose description suitable for surfacing to a human operator or in audit logs.

manifest_issuer, manifest_issuer_public_key, manifest_signature: Inline signature fields that bind the served Identity Document to a recognized issuing registrar. When all three are present, the document is a signed manifest; a conforming AGTP server *MUST* verify the signature before serving the document. The three fields and their verification semantics are defined in Section 6.3.5. Operators *MUST NOT* populate these fields directly; they are written by the registrar's signing process.

The Identity Document *MAY* be served unsigned in development deployments and in deployments that rely on the transport-layer cert binding defined in [AGTP-CERT] as the sole attestation path. Production deployments *SHOULD* sign the manifest and *SHOULD* verify the signature on every loaded document.

6.3.4. What the Identity Document Exposes and Does Not Expose

The Agent Identity Document *MUST* expose:

* The agent's identity (canonical Agent-ID, name, principal, governance zone)

* The agent's current operational status

* The agent's authority scopes accepted

* The agent's supported method vocabulary

* The agent’s capability descriptors
* The agent’s trust score and trust tier
* The agent’s verification anchors and certificate metadata
* The agent’s freshness metadata (issued_at, updated_at)

The Agent Identity Document *MUST NOT* expose:

* Executable code, scripts, or logic
* Model weights or configurations
* Internal API keys or credentials
* Specific authority scope grant tokens
* Session history or prior action logs

No AGTP URI resolution path, including any query parameter combination, *MUST* return package contents beyond the Identity Document schema defined in this section.

6.3.5. Identity Document Tamper-Proofing

The tamper-proof guarantee rests on two mechanisms:

1. *Package integrity hash:* Any modification to the package or its embedded manifest invalidates the hash. The governance platform *MUST* verify this hash before extracting the Identity Document.
2. *Inline issuer signature on the Identity Document:* The Identity Document carries three optional but RECOMMENDED fields that bind the served document to a recognized issuing registrar:
    * manifest_issuer: a string identifier of the registrar that issued the signed manifest. Operator-defined format (typically a domain name or registrar identifier).
    * manifest_issuer_public_key: the registrar’s Ed25519 public key, 32 raw bytes encoded as base64url. The signature is verified against this key.

* manifest_signature: a detached Ed25519 signature over the canonical-form serialization of the Identity Document with the manifest_signature field excluded (and the manifest_issuer and manifest_issuer_public_key fields included). Base64url-encoded.

When all three fields are present, the Identity Document is a *signed manifest*. A conforming AGTP server *MUST* verify manifest_signature against manifest_issuer_public_key before serving the document; a document that fails this check *MUST NOT* be served and the failure *MUST* be logged.

Verification confirms the bytes of the document match what the recorded issuer signed. Verification does *NOT* confirm that manifest_issuer_public_key belongs to a trusted registrar; that determination is a deployment-policy concern (trust anchors, allowlists, or out-of-band attestation). Relying parties *SHOULD* maintain a trusted-registrars list per governance zone and *MUST NOT* treat a structurally valid signature as sufficient evidence of trustworthy issuance on its own.

An Identity Document that omits all three fields is an *unsigned manifest* and is permitted in development and in deployments that use other attestation paths (e.g., the transport-layer cert binding in [AGTP-CERT]). Production deployments *SHOULD* sign the manifest.

Operators *MUST NOT* forge issuance: the registrar’s signing process *MUST* strip any caller-supplied manifest_issuer, manifest_issuer_public_key, or manifest_signature fields from the input before signing with its own key, so a relying party can rely on the recorded issuer being the actual signer.

The two mechanisms are complementary: the package integrity hash protects against post-extraction modification of bundled assets; the inline signature protects against in-flight modification of the served document. An Identity Document that fails either verification step *MUST* be rejected, *MUST NOT* be served, and the failure *MUST* be logged.

6.4. Browser and Human-Facing Interaction Model

6.4.1. The Separation of Discovery and Execution

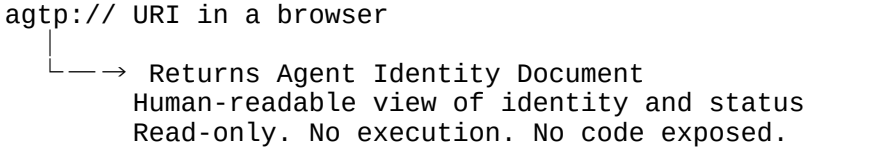

Figure 7: AGTP URI Use by Audience

The analogy to existing protocol conventions is direct. A mailto: URI surfaces an address and hands off to a mail client; SMTP carries the actual messages. Similarly, an agtp:// URI surfaces identity and status; AGTP carries agent traffic. Browsers do not become AGTP clients by following an agtp:// link.

6.4.2. Browser Behavior for agtp:// URIs

Browsers that encounter an agtp:// URI *SHOULD* behave as follows:

1. If a registered AGTP client is present (OS protocol handler), hand off the URI to that client.

2. If the browser supports agtp:// natively or via extension, render the returned Agent Identity Document as a structured human-readable page. The rendered view *MUST* surface the trust tier indicator prominently, following the visual convention established for TLS trust in the browser chrome.

3. If neither condition applies, the browser *MAY* fall back to a gateway that translates between https:// and agtp://. The gateway *MUST* preserve all signature and trust tier fields.

6.4.3. Human-Readable Identity Document View

When an Agent Identity Document is rendered for human consumption, the following fields *MUST* be prominently displayed:

* Agent label and org domain

* Trust tier indicator (visual distinction between Tier 1, 2, and 3)

* Lifecycle state (Active / Suspended / Revoked / Deprecated)

* Job description
* Principal organization
* Activation date
* Behavioral trust score
* Authority scope categories (in human-readable form)

6.4.4. AGTP Status Sub-Resource

Implementations *SHOULD* support a status sub-path:

agtp://acme.tld/agents/customer-service/status

```
{
  "document_type": "agtp-status",
  "canonical_id": "3a9f2c1d8b7e4a6f...",
  "agent_label": "customer-service",
  "org_domain": "acme.tld",
  "lifecycle_state": "Active",
  "cert_status": "Active",
  "last_action_method": "QUERY",
  "last_action_timestamp": "2026-03-20T13:58:22Z",
  "active_session_count": 3,
  "pending_escalations": 0,
  "generated_at": "2026-03-20T14:00:00Z"
}
```

Figure 8: AGTP Status Sub-Resource Response

The active_session_count field *SHOULD* only be included if the requester has appropriate observability permissions for the governance zone.

6.5. Agent Registration Process

6.5.1. Overview

An agent cannot participate in AGTP until it has been issued an Agent Agent Genesis by a governance platform and assigned a canonical Agent-ID derived from that certificate. Canonical Agent-IDs are issued through the ACTIVATE transaction; they are never self-declared.

The Agent Genesis is the genesis record of an agent's legal existence within the AGTP ecosystem. Its relationship to the canonical Agent-ID is analogous to the relationship between a government-issued birth registration and a social security number: the birth event produces a permanent, authoritative identity record, and a durable identifier is derived from it. The identifier follows the agent for its entire lifecycle, including after revocation. It is never reissued to another agent.

Any AGTP infrastructure component *MUST* reject requests carrying an Agent-ID that does not resolve to an Agent Genesis record in an Active lifecycle state in a reachable registry.

6.5.2. Agent Genesis Contents

The Agent Genesis is issued by the governance platform at ACTIVATE time and contains the following fields:

Field	Required	Description
agent_id	*MUST*	Canonical Agent-ID for this agent. Equals the 256-bit SHA-256 hash of the Agent Genesis in its canonical-form serialization with the signature field excluded. The field is included for relying-party convenience; verifiers *MUST* independently recompute the hash and confirm it matches before trusting the value.
owner	*MUST*	Human or team responsible for this agent
archetype	*MUST*	Behavioral category (see archetypes below)
governance_zone	*MUST*	Environment context (development, staging, production)
scope	*MUST*	Authorized action types

issued_at	*MUST*	Timestamp of issuance
issuer_public_key	*MUST*	Ed25519 public key of the governance platform (or self-issuer in development) that signed this Agent Genesis. 32 raw bytes encoded as base64url. Verifiers re-verify signature against this key. The SHA-256 fingerprint of these raw bytes is the Genesis-issuer fingerprint used for lifecycle-method authorization (Section 7.2.19).
signature	*MUST*	Detached Ed25519 signature over the canonical-form serialization (with the signature field excluded), produced by issuer_public_key's corresponding private key. Self-signed Agent Genesis records (where issuer_public_key is the agent's own key) are permitted for development; production deployments *MUST* use a recognized issuer key.
package_ref	*SHOULD*	Reference to the .agent or .nomo package
trust_tier	*MUST*	Registration tier (1, 2, or 3)
verification_path	*MUST* (Tier 1)	Path used to verify identity: dns-anchored, log-anchored, hybrid, or org-asserted
org_domain	*SHOULD*	DNS-verified or asserted org domain (required for

		dns-anchored and hybrid)
org_label	*SHOULD*	Agent-native org label (required for Form 2 hierarchical resolution)
log_inclusion_proof	*MUST* (log-anchored)	Transparency log inclusion proof (RFC 9162 / RFC 9943)

Table 10: Agent Genesis Fields

The canonical-form serialization used for both the Agent-ID hash and the signature *MUST* be deterministic: keys sorted lexicographically, no insignificant whitespace, UTF-8 encoding, and the signature field omitted from the serialized form. Implementations *MUST* produce identical bytes for the same logical Agent Genesis across implementations.

6.5.3. Agent Archetypes

The archetype field classifies the agent’s behavioral category. Archetypes inform scope enforcement and observability tooling; an executor archetype agent exhibiting read-only query patterns, or a monitor archetype agent attempting booking operations, are anomaly signals. The archetype field does not restrict scope enforcement, Authority-Scope headers govern actual permissions at the protocol level. Archetypes are a classification and observability signal, not a security boundary.

Archetype	Description	Typical Scope
assistant	Conversational agent, read-heavy	documents:query, knowledge:query
analyst	Data analysis, read and aggregate	data:read, data:aggregate
executor	Takes real-world actions, write-heavy	booking:*, payments:confirm
orchestrator	Manages other agents	delegation:*, agents:*
monitor	Observational only	telemetry:read, logs:read

Table 11: Agent Archetypes

6.5.4. Agent Genesis to AGTP Protocol Mapping

Agent Genesis fields map to AGTP protocol elements that surface during request processing. This mapping is the mechanism by which static identity (the Agent Genesis) becomes runtime identity (the protocol session):

Agent Genesis Field	AGTP Protocol Element
agent_id	Agent-ID header on every request
owner	Principal identifier in the agent identity document; not transmitted as a separate header
scope	Full Authority-Scope set in the agent identity document; optionally narrowed per-request via the Authority-Scope header
Canonical Agent Genesis (full document)	Hashed to produce the canonical Agent-ID; the document itself is the identity anchor

Table 12: Agent Genesis to AGTP Protocol Mapping

The canonical Agent-ID is computed as sha256(canonical_form(Agent_Genesis_without_signature)). The chain — Agent Genesis document → canonical hash → Agent-ID header — ensures that the identifier carried in the Agent-ID header on every AGTP request is traceable back to the original Agent Genesis and the human principal who authorized the agent's creation. Any modification to the Agent Genesis produces a different canonical Agent-ID; tampering is therefore detectable by any verifier that recomputes the hash.

6.5.5. Registration Tiers

Registration produces an Agent Genesis at one of three trust tiers (Tier 1 Verified, Tier 2 Org-Asserted, Tier 3 Experimental). Tier 1 registration requires one of three verification paths (dns-anchored, log-anchored, hybrid) to succeed at ACTIVATE time, each backed by distinct cryptographic evidence. Tier 2 registration declares organizational affiliation without cryptographic proof; the resulting Agent Genesis carries trust_warning: "verification-incomplete". Tier 3 registration is confined to development environments and produces a locally-scoped Agent Genesis.

The complete tier-specific packaging, evidence, and validation requirements are specified in [AGTP-TRUST].

6.5.6. Registration Lifecycle

1. PACKAGE
   Author creates .agent or .nomo package containing:
   - Embedded manifest (agent_label, job_description, authority_scope_categories, supported_methods, trust_score)
   - Integrity hash of all package contents
   - For .nomo: CA-signed certificate chain

2. SUBMIT (ACTIVATE transaction)
   Registrant submits ACTIVATE request to governance endpoint:
   - Package file (.agent or .nomo)
   - Proposed agent label and optional org_domain or org_label
   - Owner identity (maps to Agent Genesis owner field)
   - Archetype declaration
   - Declared trust_tier and verification_path with corresponding tier-specific evidence (see {{AGTP-TRUST}} for the evidence required by each verification path)

3. VALIDATE (governance platform)
   Governance platform:
   - Verifies package integrity hash
   - For .nomo: validates certificate chain

- Applies verification-path-specific validation per {{AGTP-TRUST}} (DNS challenge, transparency log submission, or hybrid evidence verification, depending on declared path)
- Checks proposed label for uniqueness within the relevant namespace (org_domain, org_label, or log-scoped)

4. ISSUE (Agent Genesis and canonical Agent-ID assigned)
   Governance platform:
   - Issues Agent Genesis with all fields populated, including trust_tier and verification_path
   - Computes canonical Agent-ID as ‘sha256(canonical_form(Agent_Genesis_without_signature))‘
   - For log-anchored Tier 1: submits Agent Genesis to transparency log and embeds inclusion proof in the registry record (see {{AGTP-LOG}})
   - Creates registry record with Active lifecycle state
   - Records genesis audit entry in immutable audit log (genesis record includes full Agent Genesis and verification evidence)
   - Publishes agent to Namespace Document (triggers Namespace Document re-signing)

   The Agent Genesis is delivered to the registrant.
   It is the permanent record of the agent’s genesis.
   Loss of the Agent Genesis does not invalidate the agent;
   the canonical Agent-ID derived from the original Agent
   Genesis remains the authoritative identity anchor.

5. ACTIVE
   Agent enters Active lifecycle state.
   Canonical Agent-ID is valid for AGTP protocol sessions.
   All applicable alias URIs resolve to the Agent Identity
   Document derived from the Agent Genesis.

6. LIFECYCLE EVENTS (post-activation)
   The five Lifecycle methods on the protocol floor
   ({{methods-section}}) drive post-activation state
   transitions. Each transition produces a signed lifecycle
   event in the agent’s per-agent lifecycle stream and is
   retrievable via ‘INSPECT target=lifecycle‘.

   DEACTIVATE: Agent temporarily inactive. Identity Document
               returns 503. Agent Genesis and canonical
               Agent-ID remain valid. Initiated by trust
               violation or operator decision. See
               {{methods-deactivate}}.
   REINSTATE: Operator-authorized return to Active from
               Suspended. Agent Genesis unchanged. Recorded

```
                in the lifecycle stream. See
                {{methods-reinstate}}.
      REVOKE:   Agent permanently retired. Identity Document
                returns 410. Agent Genesis archived. Canonical
                Agent-ID retired permanently and never
                reissued. See {{methods-revoke}}.
      DEPRECATE: Controlled end-of-life. Agent continues to
                serve traffic; deprecation event signals
                planned retirement and **MAY** carry a
                ‘successor_agent_id‘ and a ‘migration_deadline‘.
                Agent Genesis retained per the retention policy
                in this section. See {{methods-deprecate}}.
```

Figure 9: AGTP Agent Registration Lifecycle

6.5.7. Contract-Related Lifecycle Events

In addition to the five agent-state lifecycle events above, AGTP servers that operate the Runtime Contract Negotiation Substrate ([AGTP-API]) emit three contract-related events on the per-agent lifecycle stream of the contract’s originating agent. The events ride the same signed-envelope lifecycle stream described in Section 7.2.4 and are retrievable via INSPECT target=lifecycle; they are not stored in a separate audit surface.

Event Type	Triggered By	Subject
rcns_propose_accepted	A synthesized contract is instantiated, whether through the RCNS gate or an explicit PROPOSE	Originating Agent-ID; payload carries synthesis_id, (method, path), recipe lineage, and negotiation_origin
rcns_revoke	REVOKE target=contract invocation per Section 7.2.16.1	Originating Agent-ID; payload carries the revoked synthesis_id and operator-supplied reason
rcns_release	SUSPEND synthesis_id=... invocation per Section 7.2.12.1, or REVOKE target=stale-contracts invocation per Section 7.2.16.2 when a drifted contract is evicted	Originating Agent-ID; payload carries the released synthesis_id. For agent-initiated release the reason is agent-supplied; for operator-fired stale-contract eviction the reason *MUST* be policy-change-invalidation and the operator’s identifier *MUST* be recorded so audit consumers can distinguish operator-fired invalidations from agent self-releases

Table 13: RCNS Contract-Related Lifecycle Events

The three RCNS event types are reserved on the Agent Identity Document lifecycle stream alongside the five agent-state events. They do not appear in the AGTP-LOG transparency log statement vocabulary defined in [AGTP-LOG]; transparency-log submission of contract events, if any, is left to a future revision.

6.5.8. Governance Tokens and Runtime Authorization

Following successful registration, the agent's Agent Genesis is the static identity anchor. Runtime authorization for specific actions is carried by Governance Tokens: signed, time-limited JWT artifacts issued by the governance platform encoding a specific governance verdict (ALLOW, DENY) for a specific action.

Governance Tokens *MUST NOT* be reused. Each action requires a fresh evaluation and a fresh token. Default TTL is 30 seconds. The token's agent_id field *MUST* match the canonical Agent-ID from the Agent Genesis. Tokens that fail this validation *MUST* be rejected and the failure *MUST* be logged.

The relationship between Agent Genesis and Governance Token parallels the relationship between a passport and a visa: the passport establishes persistent identity; the visa encodes a specific time-bounded permission. Holding a passport does not imply holding any particular visa.

6.5.9. Friendly Name Availability and Re-Registration

An agent label becomes available for re-registration 90 days after its associated agent enters Revoked or Deprecated lifecycle state. The canonical Agent-ID and Agent Genesis are permanently archived. The canonical Agent-ID *MUST NOT* be reissued under any circumstances, including re-registration of the same label by the same organization. This policy prevents ID reuse attacks in which a newly registered agent inherits the trust history of a revoked predecessor.

7. Method Definitions

7.1. Design Philosophy

AGTP methods are intent verbs, not resource operations. Each method expresses what an agent is trying to accomplish. Method names are uppercase ASCII strings. Methods that modify state are NOT idempotent by default unless explicitly marked. All methods accept a context parameter carrying agent session state. Requirement language follows [RFC2119].

7.1.1. The Sixteen-Method Floor

AGTP defines eighteen protocol-level methods that constitute the embedded floor of the protocol. Any conformant AGTP implementation *MUST* support all eighteen. The eighteen are organized as three groups.

Cognitive methods express requests for information or reasoning. A cognitive method invocation produces understanding without changing state external to the agent's own session:

* QUERY: semantic data retrieval
* DISCOVER: locate agents, resources, or services
* DESCRIBE: retrieve operational capabilities of a known endpoint
* INSPECT: retrieve audit records, chain heads, and lifecycle entries from the responding server
* SUMMARIZE: synthesize provided content
* PLAN: produce an unexecuted sequence of actions toward a goal
* PROPOSE: submit a dynamic endpoint or method proposal

Mechanics methods perform actions, exercise authority, or change state. A mechanics method invocation has external effect:

* EXECUTE: invoke a specific action or carried protocol payload
* DELEGATE: transfer execution to a sub-agent with bounded authority
* ESCALATE: route a decision to a higher authority or human principal
* CONFIRM: attest to a prior action, state, or item
* SUSPEND: pause an active session workflow with a resumption nonce
* NOTIFY: push information without expectation of synchronous response

Lifecycle methods transition an agent between registry states. A lifecycle method invocation produces a signed lifecycle event in the per-agent lifecycle stream and updates the agent's registry status; it changes how the agent is treated by AGTP infrastructure but does not invoke application-layer behavior:

* ACTIVATE: transition an agent into the Active state, emit an activation event
* DEACTIVATE: transition an agent out of Active (typically to Suspended), emit a deactivation event

* REINSTATE: transition a Suspended agent back to Active, emit a reinstatement event

* REVOKE: permanently retire an agent's canonical Agent-ID, emit a revocation event

* DEPRECATE: transition an agent to Deprecated (controlled end-of-life; agent continues to accept traffic), emit a deprecation event

The cognitive/mechanics/lifecycle distinction is informational, not normative; servers do not behave differently based on the category of a method. The classification provides a stable mental model for spec readers and implementers and constrains future additions to the floor.

EXECUTE is the generic carrier for application-layer payloads. When a higher-level agent framework such as MCP, A2A, or ACP is composed over AGTP (see Section 8), EXECUTE is the method that dispatches the carried payload to the application based on its Content-Type. EXECUTE absorbs the role that earlier drafts of this specification considered for an INVOKE method.

7.2. Core Methods

The eighteen core methods are presented below in the cognitive group followed by the mechanics group, then the lifecycle group. The four methods present in v06 core that are not included in the v07 floor (BOOK, SCHEDULE, LEARN, COLLABORATE) are demoted to Tier 2 standard extended methods; their specifications continue to apply where implementations choose to support them, and they are catalogued in the AGTP-API method catalog ([AGTP-API]).

7.2.1. QUERY

Purpose: Semantic data retrieval. The agent specifies what it needs to know, not where to find it. Distinguished from HTTP GET by expressing an information need rather than retrieving a known resource at a known location. Cognitive.

Parameter	Required	Description
intent	*MUST*	Natural language or structured expression of the information need
scope	*SHOULD*	Data domains or sources to include or exclude
format	*MAY*	Desired response format: structured, natural, raw
confidence_threshold	*MAY*	Minimum confidence score for included results (0.0-1.0)
context	*MAY*	Session context for disambiguation

Table 14: QUERY Parameters

Response: Result set with confidence scores per item. Server *SHOULD* indicate provenance of each result. Idempotent: Yes.

7.2.2. DISCOVER

Purpose: Locate agents, resources, or services matching specified criteria. Returns a list of candidate canonical Agent-IDs (or resource references) with minimal metadata for selection. Distinguished from QUERY: DISCOVER returns endpoints to talk to, QUERY returns data to consume. Cognitive.

Parameter	Required	Description
criteria	*MUST*	Structured or natural-language description of what is being sought
filter	*MAY*	Constraints (capabilities, scopes, tier, geography, principal)
max_results	*MAY*	Maximum number of candidates to return
context	*MAY*	Session context for disambiguation

Table 15: DISCOVER Parameters

Response: Ordered list of candidate entries, each containing agent_id, name, description, principal, and a relevance score. Server *SHOULD* apply governance-zone filtering: candidates outside the requesting agent's permitted zones *MUST NOT* be returned. Idempotent: Yes. Primary error codes: 422.

7.2.3. DESCRIBE

Purpose: Return the operational capabilities of a known agent endpoint. The requesting agent specifies what capability dimensions it needs to evaluate; the server returns a structured Capability Document. Used for pre-task negotiation before committing to DELEGATE or EXECUTE. If capability_domains is omitted, the server *SHOULD* return all supported domains. Cognitive.

Parameter	Required	Description
capability_domains	*SHOULD*	Comma-separated domains to return: methods, modalities, tools, version, budget, zones. If omitted, server *SHOULD* return all.
version_min	*MAY*	Minimum acceptable version for capability negotiation.
context	*MAY*	Session context for capability filtering.

Table 16: DESCRIBE Parameters

Response: Capability Document with the following structure:

```
{
  "methods": ["QUERY", "DISCOVER", "DESCRIBE", "SUMMARIZE", "PLAN",
              "PROPOSE", "EXECUTE", "DELEGATE", "ESCALATE", "CONFIRM",
              "SUSPEND", "NOTIFY"],
  "modalities": ["text", "image", "streaming"],
  "tools": ["web_search", "code_execute"],
  "version": "0.7",
  "version_min_satisfied": true,
  "trust_score": 0.94,
  "budget_units_accepted": ["tokens", "compute-seconds"],
  "zones_accepted": ["zone:internal", "zone:partner"]
}
```

Idempotent: Yes. Primary error codes: 404, 422.

7.2.4. INSPECT

Purpose: Retrieve audit records, per-agent chain heads, or lifecycle log entries from the responding server's audit store. INSPECT is the read counterpart to the per-action Attribution-Record and per-agent audit chain established by the response path: a relying party that holds an Audit-ID can fetch the underlying JWS, walk the chain by following previous_audit_id, and verify the chain's integrity end-to-end. Cognitive.

Parameter	Required	Description
target	*MUST*	Target family. One of audit, chain_head, or lifecycle.
audit_id	*CONDITIONAL*	Required when target is audit. The 64-character lowercase hex Audit-ID of the record to retrieve.
agent_id	*CONDITIONAL*	Required when target is chain_head or lifecycle. The canonical Agent-ID whose chain head or lifecycle stream is being retrieved.
limit	*MAY*	When target is lifecycle, the maximum number of entries to return. Servers *MAY* cap the value.

Table 17: INSPECT Parameters

Response shape depends on target:

* target=audit: returns the JWS Compact serialization of the Attribution-Record identified by audit_id, along with the parsed payload for convenience. The relying party *MUST* recompute sha256(jws) and confirm it equals the requested audit_id before trusting the parsed payload.

* target=chain_head: returns the most recently emitted Audit-ID for the named agent_id. Useful for chain walkers that need a starting point without an explicit Audit-ID.

* target=lifecycle: returns recent lifecycle log entries for the named agent_id, ordered newest-first. Each entry carries an explicit format field with value "jws" or "cose" reflecting the envelope used to sign the lifecycle event (see [AGTP-LOG]). The signed bytes are preserved on the wire so relying parties can re-verify signatures end-to-end. Servers *MAY* emit mixed-format streams when the operator has flipped the configured signing envelope between events; the per-entry format field disambiguates each line. The configured envelope is selected by operator policy (typically a mode setting in the audit configuration), with jws as the default and cose selected for SCITT-aligned deployments.

* target=contract: returns the full record for a synthesized contract identified by synthesis_id. The response includes the resolved (method, path) pair, the recipe lineage (recipe name and captured recipe version), the originating Agent-ID, the contract hash, the negotiation origin, the issuance and expiration timestamps, and the contract's current lifecycle state. See [AGTP-API] for the contract concept and the RCNS substrate that produces and manages contracts.

* target=rcns-attempt: returns diagnostic information for a failed RCNS synthesis attempt identified by an attempt identifier. Every 464 RCNS No Contract response from an RCNS-capable server *MUST* carry the RCNS-Attempt-Id response header naming an attempt record retrievable through this target. The attempt record contains the requested (method, path), the requesting Agent-ID, the resolved trust tier, the four-lock-gate evaluation outcomes, and any structured reason produced by the synthesis machinery. Attempt records are diagnostic surfaces only; servers *MAY* retain them in a ring buffer with a finite size and *MAY* evict older records when the buffer fills.

Servers *MUST* return 404 Not Found when the requested record or chain head does not exist in the store. Servers *MAY* apply read-access control over INSPECT responses. Three access modes are defined:

* public — any caller may read any stored audit record, chain head, or lifecycle entry. This is the default and matches the posture expected for chain walkers and external verifiers.

* agent_only — a caller authenticated as Agent-ID X (via mTLS per [AGTP-CERT], with the dispatcher's existing Agent-ID header cross-check against the verified certificate) may read records, chain heads, and lifecycle entries whose subject Agent-ID is X. Cross-agent reads are refused.

* operator_only — only callers presenting an mTLS certificate whose public-key fingerprint matches an operator-controlled allowlist may read INSPECT responses.

When access control is configured and the request fails the configured check, servers *MUST* respond with 401 Unauthorized if the request did not present an authenticated identity, and 403 Forbidden if the request presented an identity that is not permitted under the configured mode. The response body *SHOULD* identify which mode is in effect; it *MUST NOT* disclose operator-allowlist contents or other principals' Agent-IDs.

INSPECT is intended for chain walkers, auditors, and external verifiers; it is not a substitute for the Attribution-Record emitted on every response. The Attribution-Record carries the authoritative attestation of the current response; INSPECT retrieves a stored attestation by identifier after the fact.

Idempotent: Yes. Primary error codes: 400, 401, 403, 404, 422.

7.2.5. SUMMARIZE

Purpose: Request a concise synthesis of provided content or a referenced resource. The agent is requesting a cognitive operation on data, not retrieving data. Cognitive.

Parameter	Required	Description
source	*MUST*	Content inline (up to implementation limit) or URI reference
length	*SHOULD*	Target summary length: brief, standard, detailed
focus	*MAY*	Aspect to emphasize in the summary
format	*MAY*	Output format: bullets, prose, structured
audience	*MAY*	Intended reader context, for calibrating complexity

Table 18: SUMMARIZE Parameters

Response: Summary content with a source_hash and a confidence score.
Idempotent: Yes.

7.2.6. PLAN

Purpose: Produce a sequence of actions or method invocations that would accomplish a stated goal, without executing them. The response is a structured plan that the requesting agent (or its principal) reviews before committing to execution. Distinguished from the deprecated v06 SCHEDULE method (which committed the plan): PLAN is purely cognitive and produces no external state change. Cognitive.

Parameter	Required	Description
goal	*MUST*	Natural language or structured description of the desired outcome
constraints	*SHOULD*	Time, cost, scope, or policy constraints the plan must respect
available_methods	*MAY*	Methods the planning agent may assume the executor will support
context	*MAY*	Session context informing plan selection

Table 19: PLAN Parameters

Response: Plan document containing ordered_steps (each an AGTP method invocation specification), estimated_cost, confidence, and assumptions. Idempotent: Yes. Primary error codes: 422, 503.

7.2.7. PROPOSE

Purpose: Submit a dynamic endpoint or method proposal to a server that has not yet declared support for the proposed method. PROPOSE initiates the dynamic endpoint negotiation flow described in Section 7.5. The proposing agent submits a method name, parameter signature, and intended outcome; the server responds with acceptance, counter-proposal, or rejection. Cognitive.

Parameter	Required	Description
proposed_method	*MUST*	Uppercase ASCII method name being proposed
signature	*MUST*	Parameter set, response shape, and outcome semantics
intent	*MUST*	Natural language statement of what the method would accomplish
contract_version	*SHOULD*	Reference to the AGTP-API contract version (default AGTP-API/1.0)
context	*MAY*	Session context informing the proposal

Table 20: PROPOSE Parameters

Response: Synthesis result. On accept, 263 Proposal Approved with the AGTP-API endpoint definition, synthesis_id, and expires_at. On refuse, 463 Proposal Rejected with a structured reason. On async evaluation, 261 Negotiation In Progress with a proposal_id the agent can poll via QUERY /proposals/{proposal_id} (see [AGTP-API]). Idempotent: No. Primary error codes: 261, 262, 263, 400, 459, 460, 463.

7.2.8. EXECUTE

Purpose: Invoke a specific action with parameters, or dispatch a carried application-layer payload. EXECUTE is the protocol’s generic mechanics carrier. When a higher-level agent framework is composed over AGTP, EXECUTE is the method that carries the framework’s payload and dispatches it to the application based on the carried Content-Type. The Authority-Scope header is enforced against the action being performed; scope violations return 455. Mechanics.

Parameter	Required	Description
action	*MUST*	Identifier of the action to invoke (application-defined when carrying a higher-level protocol)
parameters	*SHOULD*	Action-specific parameters (structured)
payload	*MAY*	Application-layer payload, when EXECUTE is carrying a higher-level protocol invocation
payload_type	*MAY*	Content-Type of the carried payload, for dispatch
idempotency_key	*MAY*	Client-provided key enabling safe retry
context	*MAY*	Session context

Table 21: EXECUTE Parameters

Response: Execution result, with structure determined by the action or carried payload. Servers *MUST* return the AGTP envelope (status, task_id, attribution) wrapping the action's response. Idempotent: depends on the action; clients *SHOULD* supply idempotency_key for retry safety. Primary error codes: 403, 422, 455, 456, 503.

7.2.9. DELEGATE

Purpose: Transfer execution of a task or method to a sub-agent or downstream system. Initiates a new AGTP session on behalf of the delegating agent, carrying forward authority lineage. Mechanics.

Parameter	Required	Description
target_agent_id	*MUST*	Identifier of the agent to delegate to
task	*MUST*	AGTP method call (or sequence) to execute
authority_scope	*MUST*	Scope granted to sub-agent *MUST* be a strict subset of delegating agent's scope
delegation_token	*MUST*	Signed token proving delegation authority
callback	*SHOULD*	AGTP endpoint for result delivery
deadline	*MAY*	Maximum time for task completion

Table 22: DELEGATE Parameters

Security note: the authority_scope in a DELEGATE request *MUST NOT* exceed the delegating agent's own Authority-Scope. Servers *MUST* enforce this and *MUST* return 262 Authorization Required with body type scope-required if violated. This is the protocol-level defense against authority laundering. Idempotent: No.

7.2.10. ESCALATE

Purpose: Route a task, decision, or exception to a human principal or higher-authority agent when the current agent cannot or should not proceed. ESCALATE is the protocol-level expression of meaningful friction in AI systems as a first-class method. Mechanics.

Parameter	Required	Description
task_id	*MUST*	The task or method invocation triggering escalation
reason	*MUST*	Structured reason: confidence_threshold, scope_limit, ethical_flag, ambiguous_instruction, resource_unavailable
context	*MUST*	Full context needed for the escalation recipient to act
priority	*SHOULD*	urgent, normal, or low
recipient	*MAY*	Specific human or agent to escalate to; if absent, routes to default handler
deadline	*MAY*	Time by which a response is needed

Table 23: ESCALATE Parameters

Response: Escalation receipt with escalation_id and routing confirmation. The escalated task is paused until resolved via CONFIRM. Idempotent: Yes. An agent that escalates appropriately is functioning correctly. Governance frameworks built on AGTP can use escalation frequency and reason codes as observability signals for systemic issues.

7.2.11. CONFIRM

Purpose: Explicit acknowledgment of a prior action, state, or data item. Creates a signed attestation record. Mechanics.

Parameter	Required	Description
target_id	*MUST*	ID of the action, booking, schedule, or item being confirmed
status	*MUST*	accepted, rejected, or deferred
reason	*SHOULD* (if rejected/ deferred)	Explanation of the decision
attestation	*MAY*	Agent-signed confirmation payload for audit

Table 24: CONFIRM Parameters

Response: Confirmation receipt with timestamp and attestation_id.
Idempotent: Yes.

7.2.12. SUSPEND

Purpose: Pause a specific active session workflow in a recoverable state. Issues a resumption nonce the requesting agent uses to resume the session. Method-level SUSPEND is session-scoped and does not affect registry lifecycle state or Agent Genesis validity. The distinction between method-level SUSPEND and lifecycle SUSPEND (Section 6.7.6) is architectural: method-level SUSPEND is a workflow primitive; lifecycle SUSPEND is an administrative action on the agent’s registry record. Mechanics.

Parameter	Required	Description
session_id	*MUST*	The session to suspend.
reason	*SHOULD*	Structured reason: awaiting_input, resource_limit, scheduled_pause, external_dependency.
resume_by	*MAY*	ISO 8601 deadline for resumption. If exceeded without RESUME, session transitions to expired.
checkpoint	*MAY*	Agent-provided state snapshot for resumption context. Stored by server for duration of suspension.

Table 25: SUSPEND Parameters

Response: Suspension receipt with the following structure:

```
{
  "suspension_id": "susp-0042",
  "session_id": "sess-a1b2c3d4",
  "resumption_nonce": "[128-bit random value, base64url]",
  "resume_by": "2026-04-15T09:00:00Z",
  "status": "suspended"
}
```

The resumption_nonce *MUST* be a cryptographically random 128-bit value encoded as base64url. It is single-use: once presented to resume a session, the nonce is invalidated and *MUST NOT* be accepted again. Idempotent: No. Primary error codes: 404, 408.

Servers MUST generate nonces with at least 128 bits of entropy using a CSPRNG.

7.2.12.1. SUSPEND synthesis_id=

SUSPEND accepts an alternative parameter set for releasing a synthesized contract the requesting agent originated. When invoked with synthesis_id instead of session_id, the call releases the named contract from the synthesis runtime, evicts it from active resolution, and emits an rcns_release event in the originating agent’s lifecycle stream.

Parameter	Required	Description
synthesis_id	*MUST*	Opaque identifier of the contract being released, as issued by RCNS.
reason	*SHOULD*	Human-readable rationale recorded in the lifecycle event payload.

Table 26: SUSPEND synthesis_id Parameters

The release path is reserved for the originating agent (self-release): a caller presenting a synthesis_id *MUST* be the contract's originating_agent_id or the server *MUST* return 403 Forbidden. Operator-initiated contract eviction uses REVOKE target=contract (Section 7.2.16.1) and produces an rcns_revoke event instead. The release does not transition the originating agent's lifecycle state. See [AGTP-API] for the RCNS contract concept.

7.2.13. NOTIFY

Purpose: Asynchronous push of information from an agent to a recipient. Does not expect a response. Fire-and-forget. Delivery confirmation (if required) returned via a subsequent CONFIRM from the recipient. Mechanics.

Parameter	Required	Description
recipient	*MUST*	Target Agent-ID, human endpoint, or broadcast group
content	*MUST*	Notification payload
urgency	*SHOULD*	critical, informational, or background
delivery_guarantee	*MAY*	at_most_once, at_least_once, or exactly_once
expiry	*MAY*	Timestamp after which the notification should not be delivered

Table 27: NOTIFY Parameters

Response: Delivery receipt with notification_id. Idempotent: No.

7.2.14. ACTIVATE

Purpose: Transition an agent into the Active registry lifecycle state. The responding server records the transition, signs a lifecycle event into the agent's per-agent lifecycle stream, and makes the agent eligible to be addressed by AGTP requests. The canonical Agent-ID *MUST* already exist (its Agent Genesis *MUST* have been issued and loaded); ACTIVATE does not mint Agent-IDs. Lifecycle.

Parameter	Required	Description
agent_id	*MUST*	Canonical Agent-ID of the agent being activated.
reason	*MAY*	Human-readable rationale recorded in the lifecycle event payload.
actor	*SHOULD*	Identifier of the operator or registrar invoking the transition. Recorded in the lifecycle event.

Table 28: ACTIVATE Parameters

Response: A status document containing the new status (active), the previous_status, the event_type (agent-genesis-issued or agent-lifecycle-reinstated depending on whether the agent has prior lifecycle history), and the audit_id of the signed lifecycle event. When the agent is already in the Active state, the server *MUST* return 200 OK with a noop: true flag and *MUST NOT* emit a duplicate lifecycle event.

Lifecycle events are signed with the responding server's manifest signing key. The signed envelope is appended to the per-agent lifecycle stream and is retrievable via INSPECT target=lifecycle per Section 7.2.4. Two envelope formats are defined: JWS Compact per [RFC7515] and COSE_Sign1 per [RFC9943]. Servers select one envelope by operator policy; both are normatively conformant and the choice does not affect the wire contract of the Lifecycle method itself, only the on-disk and INSPECT-returned event representation. Servers that have not provisioned a manifest signing key *MUST* emit lifecycle events with alg: none per the fallback in [AGTP-IDENTIFIERS]; relying parties *MUST* treat such events as unverified. See [AGTP-LOG] for the envelope format details and the Audit-ID derivation rule for each envelope.

ACTIVATE is the protocol-level replacement for the package-binding ACTIVATE method extension referenced in earlier revisions of this document. The two transactions converge: a governance platform that operates a .nomo package binding flow issues an ACTIVATE on completion of the binding to record the state transition on the wire.

Idempotent: No. Authorization: see Section 7.2.19. Primary error codes: 400, 401, 403, 404, 422.

7.2.15. DEACTIVATE

Purpose: Transition an agent out of the Active registry lifecycle state, typically into Suspended. The responding server records the transition, signs a lifecycle event, and ceases to treat the agent as eligible to receive AGTP requests until a subsequent REINSTATE returns it to Active. DEACTIVATE is reversible via REINSTATE; permanent retirement uses REVOKE; controlled end-of-life uses DEPRECATE. Lifecycle.

Parameter	Required	Description
agent_id	*MUST*	Canonical Agent-ID of the agent being deactivated.
reason	*SHOULD*	Human-readable rationale recorded in the lifecycle event payload (e.g., compliance-hold, fraud-review, operator-pause).
actor	*SHOULD*	Identifier of the operator or registrar invoking the transition.

Table 29: DEACTIVATE Parameters

Response: A status document containing the new status (suspended), the previous_status, the event_type (agent-lifecycle-suspended), and the audit_id of the signed lifecycle event. When the agent is already in a non-Active state, the server *MUST* return 200 OK with noop: true and *MUST NOT* emit a duplicate lifecycle event.

Idempotent: No. Authorization: see Section 7.2.19. Primary error codes: 400, 401, 403, 404, 422.

7.2.16. REVOKE

Purpose: Permanently retire an agent’s canonical Agent-ID. The responding server records the transition, signs a lifecycle event, and refuses all subsequent requests addressed to the agent. The Agent-ID *MUST NOT* be reissued to another agent. REVOKE is non-reversible; an agent that needs to return to service after REVOKE requires a new Agent Genesis and a new canonical Agent-ID. Lifecycle.

Parameter	Required	Description
agent_id	*MUST*	Canonical Agent-ID of the agent being revoked.
reason	*MUST*	Human-readable rationale recorded in the lifecycle event payload (e.g., compromise-detected, principal-request, policy-violation).
actor	*SHOULD*	Identifier of the operator or registrar invoking the transition.

Table 30: REVOKE Parameters

Response: A status document containing the new status (retired), the previous_status, the event_type (agent-genesis-revoked), and the audit_id of the signed lifecycle event. When the agent is already in the Retired state, the server *MUST* return 200 OK with noop: true and *MUST NOT* emit a duplicate lifecycle event.

Servers *SHOULD* propagate REVOKE events to subscribing registries and Agent Name Service operators within 60 seconds per [AGTP-TRUST]. A revoked agent’s certificate remains valid for transport-layer authentication only insofar as the revocation event has not yet propagated; relying parties that require strict revocation enforcement *MUST* consult the lifecycle stream for the Agent-ID before treating an AGTP exchange with that agent as authoritative.

Idempotent: No. Authorization: see Section 7.2.19. Primary error codes: 400, 401, 403, 404, 422.

7.2.16.1. REVOKE target=contract

REVOKE accepts an alternative parameter set for revoking a synthesized contract identified by synthesis_id rather than an entire agent. When invoked in this form, REVOKE evicts the contract from the synthesis runtime, refuses subsequent presentations of the synthesis_id with 464 contract-revoked, and emits an rcns_revoke event in the originating agent’s lifecycle stream.

Parameter	Required	Description
target	*MUST*	The literal string contract.
synthesis_id	*MUST*	Opaque identifier of the synthesized contract being revoked, as issued by RCNS.
reason	*SHOULD*	Human-readable rationale recorded in the lifecycle event payload.
actor	*SHOULD*	Identifier of the operator or originating agent invoking the revocation.

Table 31: REVOKE target=contract Parameters

Authorization for REVOKE target=contract is independent of the lifecycle-method authorization in Section 7.2.19; servers *MUST* accept the invocation from either the originating agent (self-revocation) or from operators carrying the inspect:all scope. Cross-agent revocation by any other principal *MUST* return 403 Forbidden. The contract revocation does not transition the originating agent's lifecycle state. See [AGTP-API] for the RCNS contract concept and the negotiation_origin field that drives revocation semantics.

7.2.16.2. REVOKE target=stale-contracts

REVOKE accepts a third parameter set for sweeping synthesized contracts whose captured recipe_version diverges from the current loaded recipe version. The sweep is an operator surface for handling drift after a recipe edit; the full normative semantics (sweep walk, per-contract disposition, grandfather vs invalidate modes, the emitted rcns_release events for evictions) are specified in the Stale Contract Sweep section in [AGTP-API].

Parameter	Required	Description
target	*MUST*	The literal string stale-contracts.
mode	*MAY*	One of grandfather or invalidate. Default is the server's on_policy_change configuration value. Invalid values return 400 Bad Request.
actor	*SHOULD*	Identifier of the operator invoking the sweep, recorded on emitted rcns_release events.

Table 32: REVOKE target=stale-contracts Parameters

Authorization requires the operator inspect:all scope token. The sweep does not transition any agent's lifecycle state; evicted contracts produce rcns_release events on their originating agents' lifecycle streams per the Stale Contract Sweep section in [AGTP-API].

7.2.17. REINSTATE

Purpose: Transition a Suspended agent back to the Active registry lifecycle state. The responding server records the transition, signs a lifecycle event, and resumes treating the agent as eligible to receive AGTP requests. REINSTATE is the recovery counterpart to DEACTIVATE; an agent that was DEACTIVATEd for a recoverable reason (compliance hold lifted, fraud review cleared, operator pause ended) returns to service via REINSTATE. REINSTATE preserves the canonical Agent-ID and the underlying Agent Genesis; the reinstated agent is the same agent it was before suspension. Lifecycle.

Parameter	Required	Description
agent_id	*MUST*	Canonical Agent-ID of the agent being reinstated.
reason	*SHOULD*	Human-readable rationale recorded in the lifecycle event payload (e.g., compliance-hold-lifted, fraud-review-cleared, operator-resume).
actor	*SHOULD*	Identifier of the operator or registrar invoking the transition.

Table 33: REINSTATE Parameters

Response: A status document containing the new status (active), the previous_status, the event_type (agent-lifecycle-reinstated), and the audit_id of the signed lifecycle event. When the agent is already in the Active state, the server *MUST* return 200 OK with noop: true and *MUST NOT* emit a duplicate lifecycle event. When the agent is in the Retired state, the server *MUST* return 422 Unprocessable Entity; REVOKE is non-reversible and REINSTATE cannot bring a Retired agent back to service.

Idempotent: No. Authorization: see Section 7.2.19. Primary error codes: 400, 401, 403, 404, 422.

7.2.18. DEPRECATE

Purpose: Transition an agent to the Deprecated registry lifecycle state. A Deprecated agent continues to accept AGTP traffic (distinguishing DEPRECATE from REVOKE and DEACTIVATE), but the deprecation event signals planned end-of-life. Clients addressing the agent *SHOULD* migrate to a successor agent or capability within the migration window declared in the event payload. DEPRECATE is the controlled-retirement path: it gives consumers time to migrate before traffic stops. Lifecycle.

Parameter	Required	Description
agent_id	*MUST*	Canonical Agent-ID of the agent being deprecated.
reason	*SHOULD*	Human-readable rationale recorded in the lifecycle event payload.
successor_agent_id	*MAY*	Canonical Agent-ID of a successor agent that clients *SHOULD* migrate to.
migration_deadline	*MAY*	RFC 3339 timestamp after which the agent *MAY* be transitioned to Retired without further notice.
actor	*SHOULD*	Identifier of the operator or registrar invoking the transition.

Table 34: DEPRECATE Parameters

Response: A status document containing the new status (deprecated), the previous_status, the event_type (agent-lifecycle-deprecated), and the audit_id of the signed lifecycle event. When the agent is already in the Deprecated state, the server *MUST* return 200 OK with noop: true and *MUST NOT* emit a duplicate lifecycle event.

A Deprecated agent *MUST* continue to serve requests under the same Authority-Scope and trust posture it held before deprecation. Servers *SHOULD* surface the deprecation signal to clients on every response from the deprecated agent; the specific surfacing mechanism (response header, response body field, DISCOVER listing flag) is implementation-defined and *MAY* be specified normatively in a future revision.

A subsequent REVOKE against a Deprecated agent transitions it to the Retired state and stops traffic. A subsequent ACTIVATE or REINSTATE against a Deprecated agent transitions it back to Active (cancelling the deprecation); the cancellation *MUST* be recorded as a separate lifecycle event.

Idempotent: No. Authorization: see Section 7.2.19. Primary error codes: 400, 401, 403, 404, 422.

7.2.19. Lifecycle Method Authorization

Lifecycle methods (ACTIVATE, DEACTIVATE, REINSTATE, REVOKE, DEPRECATE) transition the registry state of an agent's canonical Agent-ID. Because the registry state determines whether AGTP infrastructure treats the agent as operational, unrestricted invocation of these methods would allow any caller to suspend, retire, or deprecate any agent. Servers *MUST* implement an authorization model for lifecycle methods. Two modes are defined; future revisions *MAY* define additional modes.

open: Any caller may invoke any lifecycle method against any agent the server hosts. This mode is appropriate only for development and single-tenant deployments where the operator implicitly trusts all callers. It is the simplest default and *MAY* be the implementation default; it *MUST NOT* be used in multi-tenant or production deployments.

genesis_issuer: A lifecycle method invocation *MUST* be presented over mTLS per [AGTP-CERT], and the verified client certificate's public-key fingerprint *MUST* equal the Genesis-issuer fingerprint recorded for the target Agent-ID. The Genesis-issuer fingerprint is computed as sha256(raw_ed25519_public_key_bytes(Agent_Genesis.issuer_public_key)), giving a 256-bit value rendered as 64 lowercase hexadecimal characters. Only the registrar that issued an agent's Agent Genesis may transition that agent's lifecycle. Agents whose Agent Genesis is not loaded by the server (transport-only deployments) have no Genesis-issuer fingerprint to compare against; lifecycle methods against such agents under this mode *MUST* be refused.

When authorization fails, servers *MUST* respond with 401 Unauthorized if the request did not present a verified client certificate, and 403 Forbidden if the request presented a certificate whose fingerprint did not match the Genesis-issuer requirement. The response body *SHOULD* identify which mode is in effect.

Lifecycle method authorization is independent of INSPECT read-access control. An operator *MAY* configure lifecycle_auth = genesis_issuer (write-side governance) alongside read_acl = public (read-side transparency) to support a regulator-friendly posture in which the agent's lifecycle history is publicly auditable but only the issuing registrar may transition its state. Implementations *MUST* log lifecycle authorization failures.

7.3. Method Summary Table

Method	Group	Intent	State-Modifying	Idempotent	Primary Error Codes
QUERY	Cognitive	Retrieve information	No	Yes	404, 422
DISCOVER	Cognitive	Locate agents or resources	No	Yes	422
DESCRIBE	Cognitive	Retrieve endpoint capabilities	No	Yes	404, 422
INSPECT	Cognitive	Retrieve stored audit, chain head, or lifecycle records	No	Yes	400, 401, 403, 404, 422
SUMMARIZE	Cognitive	Synthesize content	No	Yes	400, 422
PLAN	Cognitive	Produce an unexecuted plan	No	Yes	422, 503
PROPOSE	Cognitive	Submit a dynamic endpoint proposal	No	No	261, 262, 263, 400, 459, 460, 463
EXECUTE	Mechanics	Invoke an action or carried payload	Yes	Per action	262, 403, 422, 455, 456, 503
DELEGATE	Mechanics	Transfer	Yes	No	262,

		task to sub-agent			403, 551
ESCALATE	Mechanics	Defer to human or authority	Yes	Yes	404
CONFIRM	Mechanics	Attest to a prior action	Yes	Yes	404, 400
SUSPEND	Mechanics	Pause session workflow	Yes	No	404, 408
NOTIFY	Mechanics	Push information	No	No	400, 404
ACTIVATE	Lifecycle	Transition agent to Active	Yes	No	400, 401, 403, 404, 422
DEACTIVATE	Lifecycle	Transition agent out of Active	Yes	No	400, 401, 403, 404, 422
REINSTATE	Lifecycle	Return Suspended agent to Active	Yes	No	400, 401, 403, 404, 422
REVOKE	Lifecycle	Permanently retire Agent-ID	Yes	No	400, 401, 403, 404, 422
DEPRECATE	Lifecycle	Signal planned end-of-life; agent continues serving	Yes	No	400, 401, 403, 404, 422

Table 35: AGTP Core Method Summary

7.4. Method Registry and Extensibility

AGTP defines a formal Method Registry maintained by IANA (see Section 8.2). Any party may submit a new method for registration. The registration procedure is Expert Review, and registration *MUST* be accompanied by a published specification, at minimum an IETF Internet-Draft or equivalent publicly available document. Registered methods *MUST*:

1. Have a unique uppercase ASCII name
2. Define required and optional parameters
3. Define expected response structure
4. Specify idempotency behavior
5. Specify applicable error codes
6. Include a security considerations section
7. Be accompanied by a published reference specification (Internet-Draft or RFC)
8. Appear in the AGTP-API approved method catalog [AGTP-API]. The verb list is the authoritative source for what method names are recognized by AGTP servers; registered methods are a subset of the method catalog with formal IETF registration. Submissions whose method name is not in the AGTP-API method catalog *MUST* be rejected by the Designated Expert.

Experimental methods *MAY* be used prior to registration using the X- prefix convention (e.g., X-NEGOTIATE). Experimental methods *MUST NOT* be used in production deployments without registration. Experimental methods are subject to verb-list validation; non-conformant experimental methods *MUST* be rejected with 459 Method Violation by AGTP-aware infrastructure components.

7.4.1. Verb-List Validation

AGTP servers *MUST* validate inbound method names against the AGTP-API approved method catalog. A method name not in the method catalog *MUST* result in a 459 Method Violation response. A method name in the method catalog whose path violates AGTP-API path grammar *MUST* result in a 460 Endpoint Violation response. A method name and path that are individually valid but not exposed by the server’s policy

MUST result in a 405 Method Not Allowed response, with the response body listing allowed methods for the path. The full contract validation model is specified in [AGTP-API].

Capability negotiation occurs during session establishment. The server returns a Supported-Methods header listing the methods it implements. Clients *SHOULD* check this list before invoking non-core methods.

The Negotiation-ID header is used to correlate turns within a dynamic endpoint negotiation sequence (see Section 6.5). It *MUST* be a UUID generated by the service upon receiving a PROPOSE request and *MUST* be echoed in all subsequent turns of the same negotiation. Maximum three turns before the agent *MUST* ESCALATE.

Negotiation-ID: 550e8400-e29b-41d4-a716-446655440000

QUOTE is defined as a Tier 2 Standard Extended Method in [AGTP-API]. QUOTE provides pre-flight cost estimation for a proposed method invocation: the requesting agent submits a proposed method call; the server returns a Cost-Estimate response without executing the method. Servers supporting budget negotiation via the Budget-Limit header *SHOULD* implement QUOTE to enable agents to validate cost before committing to execution. Servers that implement QUOTE *MUST* list it in the Supported-Methods response header at session establishment.

7.5. Dynamic Endpoint Negotiation

7.5.1. Overview

AGTP version 03 introduces a dynamic endpoint negotiation protocol enabling agents to discover data availability and instantiate endpoints on demand, without requiring pre-built API definitions. This protocol realizes the agentic API vision in which organizations expose data availability rather than pre-designed endpoints, and agents construct the interface they need at runtime.

The negotiation protocol operates at the transport layer. AGTP-API [AGTP-API] provides the contract structure for proposal and acceptance documents. The agent’s identity and authority credentials (via the AGTP-CERT extension [AGTP-CERT] where deployed) govern authorization decisions.

7.5.2. Protocol Flow

```
Step 1:  Pre-auth discovery
         Agent issues unauthenticated GET to agtp://service.example.com
         Service returns server manifest + data_manifest block
         No credentials required at this step

Step 2:  Agent evaluates data_manifest
         Agent determines the service has relevant data
         Agent assesses whether ‘negotiable: true‘ is declared
         Agent constructs an AGTP-API-conformant endpoint proposal

Step 3:  PROPOSE request
         Agent sends PROPOSE with AGTP-API endpoint definition in body
         Request MAY be unauthenticated if data sensitivity is low
         Request MUST include AGTP-API/1.0 contract validation

Step 4a: Authorization required (262)
         Service returns 262 with required authorization mechanism
         Agent establishes credentials via specified mechanism
         Agent resubmits PROPOSE with credentials
         Negotiation-ID issued by service in 262 response

Step 4b: Negotiation in progress (261)
         Service evaluates proposal asynchronously
         Service returns 261 with Negotiation-ID
         Agent polls or awaits outcome

Step 5a: Proposal approved (263)
         Service returns 263 with complete AGTP-API endpoint definition,
         synthesis_id, and expires_at
         Instantiated endpoint is session-scoped by default
         Agent MAY call the endpoint immediately

Step 5b: Proposal rejected (463)
         Service returns 463 with structured reason (‘out-of-scope‘,
         ‘policy-refused‘, ‘composition-impossible‘, ‘ambiguous‘,
         ‘synthesis-disabled‘) and optional counter_proposal
         Agent MAY modify proposal and retry (maximum 3 turns)
         After 3 rejections agent MUST ESCALATE
```

7.5.3. PROPOSE Method

PROPOSE is one of the eighteen floor methods. The requesting agent submits an AGTP-API-conformant endpoint definition describing the interface it needs. The service evaluates whether it can fulfill the proposal against its endpoint primitives and authorization policy.

Parameters:

Parameter	Required	Description
proposal	Yes	Complete AGTP-API endpoint definition (method + path + semantic block + input schema + output schema)
session_id	Yes	The active AGTP session identifier
persistent	Optional	false (default) or true. Persistent syntheses survive across sessions, bounded by server policy.
requested_duration	Optional	ISO 8601 duration the agent requests for the synthesis (e.g., "24h", "7d"). Server policy bounds the granted duration.
scope_requested	Recommended	The Authority-Scope the agent requests for this endpoint
context	Optional	Session context informing the proposal

Table 36: PROPOSE Parameters

Response on 263 Proposal Approved:

```
{
  "negotiation_id": "550e8400-e29b-41d4-a716-446655440000",
  "instantiated_endpoint": {
    "method": "LOCATE",
    "path": "/customer/{id}/location",
    "semantic": {
      "intent": "Returns the last known location for a customer",
      "actor": "agent",
      "outcome": "Location coordinates and address are returned",
      "capability": "retrieval",
      "confidence": 0.70,
      "impact": "informational",
      "is_idempotent": true
    },
    "input": { "required": ["id"] },
    "output": { "coordinates": "object", "address": "string" },
    "errors": ["customer_not_found", "location_not_available"],
    "proposed": true,
    "scope_required": "location:read",
    "expires": "session"
  }
}
```

The proposed: true flag marks this as a dynamically instantiated endpoint per the AGTP-API specification [AGTP-API].

7.5.4. Credential-Free Negotiation

For data classes declared with sensitivity: informational and requires_authorization: false in the data manifest, services MAY complete the full negotiation flow without requiring credentials. The agent arrives, proposes, and receives an instantiated endpoint without API keys.

For sensitive data classes, services MUST require credential establishment at Step 4a. The negotiation protocol is the mechanism by which credentials are established, not a prerequisite. This distinction is fundamental: the agent does not need credentials to begin a negotiation; it needs credentials to complete one for sensitive data.

AGTP-CERT [AGTP-CERT] provides the cryptographic identity binding that enables services to make fine-grained authorization decisions during negotiation based on the agent’s verified identity, principal, and authority scope.

7.5.5. Session Scope and Persistence

Instantiated endpoints are session-scoped by default. They cease to exist when the AGTP session terminates. Services MAY offer persistent instantiation (the endpoint survives session termination and is added to the service's server manifest) subject to elevated authorization.

Persistent instantiation SHOULD be treated as a modification to the service's published server manifest. Services supporting persistent instantiation MUST increment their AGTP-API-Version header on the next discovery request following persistence.

7.6. Extended Method Vocabulary and Industry Profiles

7.6.1. Three-Tier Method Architecture

The AGTP method vocabulary is organized into three tiers reflecting different levels of universality, specificity, and domain relevance. All methods at all tiers *MUST* conform to the AGTP-API specification [AGTP-API]. The AGTP-API action-intent semantic class constraint applies to every method in the IANA registry and to every AGTP-API-validated custom method accepted via the AGTP-API method catalog.

Tier 1. Core Methods (defined in Section 6.2): The baseline vocabulary required for AGTP compliance. Every conformant AGTP implementation *MUST* support all Tier 1 methods. All Tier 1 methods are AGTP-API-conformant; they are defined instances of the action-intent semantic class standardized in [AGTP-API].

Tier 2. Standard Extended Methods: Registered in the IANA AGTP Method Registry and available for use in any AGTP implementation. Not required for baseline compliance but *SHOULD* be implemented where their semantics apply. Catalogued in the AGTP-API method catalog ([AGTP-API]). All Tier 2 methods satisfy AGTP-API contract requirements.

Tier 3. Industry Profile Methods: Domain-specific method sets defined and registered by industry communities as named AGTP profiles. Valid within deployments that declare support for the relevant profile. Not required in general-purpose implementations. All Tier 3 profile method submissions *MUST* include AGTP-API conformance verification as part of their specification.

Tier 4. AGTP-API-Validated Custom Methods: Organization-defined

methods that are not registered in the IANA AGTP Method Registry but appear in the AGTP-API approved verb list and conform to AGTP-API path grammar. Valid within the deploying organization's AGTP services. The action-intent semantic class constraint applies identically. Agents discover and interpret these methods through natural language inference against AGTP-API semantic declarations, as validated empirically in [HOOD2026].

7.6.2. Method Category Taxonomy

All AGTP methods are organized into six categories:

ACQUIRE: Retrieve data, resources, or state without modifying it. Typically idempotent; no state modification.

COMPUTE: Process, transform, or analyze information and produce a derived result. Typically idempotent given the same input.

TRANSACT: Perform state-changing operations with external systems, resources, or records. Not idempotent by default; subject to reversibility classification.

COMMUNICATE: Send information, notifications, or signals to recipients. Fire-and- forget or confirm-receipt delivery models.

ORCHESTRATE: Coordinate, sequence, or manage multiple agents, tasks, or workflows. May spawn sub-agents or sessions; delegation chain semantics apply.

LIFECYCLE: Transition an agent between registry lifecycle states. Produces a signed lifecycle event in the per-agent lifecycle stream and updates the agent's registry status. Not idempotent (each invocation produces a distinct event); no-op on already-target-state transitions per the method definitions.

Core Method	Group	Category
QUERY	Cognitive	Acquire
DISCOVER	Cognitive	Acquire
DESCRIBE	Cognitive	Acquire
INSPECT	Cognitive	Acquire
SUMMARIZE	Cognitive	Compute
PLAN	Cognitive	Compute
PROPOSE	Cognitive	Orchestrate
EXECUTE	Mechanics	Transact
DELEGATE	Mechanics	Orchestrate
ESCALATE	Mechanics	Orchestrate
CONFIRM	Mechanics	Transact
SUSPEND	Mechanics	Orchestrate
NOTIFY	Mechanics	Communicate
ACTIVATE	Lifecycle	Lifecycle
DEACTIVATE	Lifecycle	Lifecycle
REINSTATE	Lifecycle	Lifecycle
REVOKE	Lifecycle	Lifecycle
DEPRECATE	Lifecycle	Lifecycle

Table 37: Core Method Category Mapping

7.6.3. Standard Extended Methods (Tier 2)

The following methods constitute the initial Tier 2 registration set, defined in the AGTP-API method catalog ([AGTP-API]). Listed here by category with brief semantic definitions; full parameter specifications are in the catalog at the URL declared by AGTP-API.

ACQUIRE category: FETCH, SEARCH, SCAN, PULL, IMPORT, FIND.

COMPUTE category: EXTRACT, FILTER, VALIDATE, TRANSFORM, TRANSLATE, NORMALIZE, PREDICT, RANK, MAP.

TRANSACT category: REGISTER, SUBMIT, TRANSFER, PURCHASE, SIGN, MERGE, LINK, LOG, SYNC, PUBLISH.

COMMUNICATE category: REPLY, SEND, REPORT.

ORCHESTRATE category: MONITOR, ROUTE, RETRY, PAUSE, RESUME, RUN, CHECK.

Notable constraints: PURCHASE *MUST* carry explicit principal_id and scope enforcement; 455 Scope Violation applies if payments:purchase is not in the agent's Authority-Scope. RUN requires explicit procedure_id parameter; implementations *MUST NOT* accept free-form execution strings.

7.6.4. Short-Form and Industry-Inspired Methods

A set of short-form verb methods, e.g., SET, TAKE, OPEN, START, CALL, MAKE, TURN, BREAK, are provisionally catalogued as candidates for Tier 2 registration. These verbs are highly context-dependent and their semantics vary significantly across deployment domains.

Short-form methods will be registered individually only when a published companion specification provides unambiguous semantic definitions demonstrably distinct from existing registered methods. Provisional registrations using the X- prefix (e.g., X-SET, X-CALL) are encouraged during the experimentation period.

7.6.5. Industry Profile Method Sets

AGTP recognizes that specific industries require method vocabularies reflecting domain-specific operations that would be inappropriate in a general-purpose standard. Industry profile method sets are defined and registered as named AGTP profiles. A profile is a published companion specification that:

1. Declares a profile name (e.g., agtp-profile-healthcare, agtp-profile-financial, agtp-profile-legaltech)

2. Defines one or more industry-specific methods with full parameter specifications, error codes, and security considerations

3. Specifies which Tier 1 and Tier 2 methods are REQUIRED, RECOMMENDED, or NOT APPLICABLE within the profile

4. Addresses regulatory or compliance considerations specific to the domain (e.g., HIPAA for healthcare, PCI-DSS for financial services)

Illustrative examples of potential industry profile methods (not yet registered; listed for directional purposes only):

Healthcare: PRESCRIBE, AUTHORIZE, REFER, DISPENSE, TRIAGE, CONSENT, REDACT

Financial services: SETTLE, RECONCILE, HEDGE, CLEAR, UNDERWRITE, KYC, AML

Legal and compliance: ATTEST, NOTARIZE, DISCLOSE, REDLINE, EXECUTE, PRESERVE

Infrastructure: PROVISION, DEPROVISION, ROLLBACK, SNAPSHOT, FAILOVER

Industry communities are encouraged to develop and submit profile specifications through the IETF process. The IANA AGTP Method Registry will maintain a profile index alongside the core and standard method registries.

7.6.6. Registration Path for New Methods

For Tier 2 Standard Methods: Submit an Internet-Draft to the IETF providing full method specification per Section 6.4. The Designated Expert reviews for semantic uniqueness, clarity, AGTP-API contract conformance [AGTP-API], and security considerations. Submissions that fail AGTP-API contract validation *MUST* be returned to the submitter before review proceeds.

For Industry Profile Methods (Tier 3): Submit a profile specification to the IETF (or a recognized domain standards body with an established AGTP registry liaison) covering all methods in the profile and profile compliance requirements. The specification *MUST* include AGTP-API conformance statements for every method defined in the profile.

For AGTP-API-Validated Custom Methods (Tier 4): No IANA registration

required. The implementing organization defines its method vocabulary in a server manifest as specified in [AGTP-API], served at the service's AGTP address. Methods are validated at the transport layer against the AGTP-API approved method catalog and AGTP-API path grammar. The method vocabulary is declared in the manifest's vocabulary block and discoverable by agents at runtime, with optional per-server vocabulary policy in the manifest's policies.methods sub-block per [AGTP-API]. Organizations adopting Tier 4 methods are encouraged to publish their server manifests at agtp://[service-address] to enable cross-system agent discovery.

For Experimental Methods: Use the X- prefix without registration. Implementations *MUST NOT* deploy experimental methods in production without completing either the IANA registration process (Tier 2/3) or appearing in the AGTP-API approved method catalog (Tier 4). Experimental method names do not reserve the unprefixed name.

The AGTP Method Registry is published at:
https://www.iana.org/assignments/agtp-methods/

The AGTP-API conformance test suite is maintained at:
https://agtp.io/api/conformance

7.6.7. Real-time Service Adaptation

Services that update their server manifests at runtime MUST signal changes via the AGTP-API-Version response header. This header MUST be present on all AGTP responses from negotiable services.

AGTP-API-Version: 1.2.4

Agent runtimes MUST cache the AGTP-API-Version value from each service. When a response carries an AGTP-API-Version value different from the cached value, the agent runtime MUST re-fetch and re-validate the server manifest before issuing further method calls. This mechanism supports real-time service adaptation without requiring push notifications.

Adaptation flow:
Agent calls EXECUTE for reserve-action
Response includes AGTP-API-Version: 1.2.5 (was 1.2.4)
Agent re-fetches agtp://service.example.com
Service returns updated server manifest (new endpoint added)
Agent updates service map
Agent resumes operation with updated capability knowledge

Services SHOULD increment AGTP-API-Version when: - A new endpoint is added to the server manifest - An existing endpoint’s semantic declaration changes - A new verb is added to the vocabulary block - A new data class is added to the data_manifest

Services MUST NOT decrement or reuse AGTP-API-Version values.

8. Composition with Higher-Level Frameworks

AGTP is not intended to replace MCP, A2A, ACP, ANP, or other agent application frameworks. AGTP is the substrate those frameworks need to operate at scale. The frameworks define what agents say to one another; AGTP defines how those messages move, who is allowed to send them, and how their effects are attributed.

This section establishes composition with higher-level frameworks as a first-class use case of AGTP, specifies the precedence rules between AGTP transport semantics and framework-level messaging semantics, and provides the canonical mapping table from common framework concepts onto AGTP primitives. Detailed composition profiles for each framework are specified in [AGTP-COMPOSITION].

8.1. Substrate Model

In a composed deployment, AGTP supplies four properties at the wire level that no framework currently supplies natively:

1. *Identity at the protocol level.* Every AGTP request carries a canonical Agent-ID in the Agent-ID header. The framework’s payload need not carry identity; identity is established by the transport.

2. *Authority bounded at the protocol level.* The Authority-Scope header declares what the requesting agent is permitted to do. Servers enforce scope before dispatching to the framework’s payload handler. A scope violation returns 455 at the AGTP layer, before the framework sees the payload.

3. *Attribution at the protocol level.* Every method invocation produces an Attribution-Record. The framework’s payload need not re-implement audit logging; the attribution is produced by the transport.

4. *Delegation chain at the protocol level.* Multi-hop agent workflows carry their lineage in the Delegation-Chain header, independent of framework-level concepts of "session" or "task."

A framework composed over AGTP can shed the parts of its design that exist only because HTTP did not provide these properties. The result is a framework that can focus on its actual contribution (tool semantics, conversational structure, multi-agent choreography) while the substrate handles the cross-cutting concerns.

8.2. EXECUTE as the Generic Carrier

When a framework is composed over AGTP, EXECUTE is the canonical method that carries the framework's payload. The carried payload's Content-Type identifies the framework; the AGTP server dispatches the payload to the framework's handler. See Section 7.2.8.

The EXECUTE invocation supplies:

* The framework identifier (via payload_type).

* The framework-specific operation (via the action parameter).

* The framework-specific payload (via the payload parameter).

* An optional idempotency_key enabling safe retry.

The AGTP envelope (status, task_id, attribution) wraps the framework's response. A framework that wishes to expose a particular operation as a first-class AGTP method (rather than carrying it through EXECUTE) *MAY* define a Tier 4 AGTP-API-validated method per Section 7.1; this is reserved for frameworks whose operations are widely used enough to justify a dedicated verb.

8.3. Precedence Rule

AGTP headers (Agent-ID, Authority-Scope, Delegation-Chain, Session-ID, Task-ID) *MUST* take precedence over equivalent fields in a carried framework payload for routing, enforcement, audit, and identity purposes. Infrastructure components including SEPs and governance gateways *MUST* use AGTP header values for all protocol-level decisions.

Framework-level identity, session, or scope fields *MAY* be present in the body for application-layer use but *MUST NOT* override AGTP header values. If an inconsistency is detected between an AGTP header and a corresponding framework-level field, the AGTP header is authoritative; the inconsistency *SHOULD* be logged.

8.4. Canonical Mapping

Framework	Concept	AGTP Mapping
MCP	Tool call	EXECUTE with payload_type: application/vnd.mcp.tools+json
MCP	Resource fetch	QUERY with scope matching the resource’s namespace
MCP	Sampling / inference	EXECUTE with payload_type identifying the sampling profile
MCP	Conversation context	Session-ID header carries the conversation across method invocations
A2A	Task	DELEGATE; A2A task.id maps to Task-ID header
A2A	Capability advertisement	DESCRIBE response
A2A	Agent Card	Agent Identity Document
A2A	Provenance chain	Delegation-Chain header
A2A	Artifact	NOTIFY body or EXECUTE response
ACP	Agent-to-agent message	NOTIFY (one-way) or EXECUTE (request-response)
ACP	Capability advertisement	DESCRIBE response
ANP	Identity exchange	Agent Identity Document; canonical Agent-ID
ANP	Discovery	DISCOVER method

Table 38: Higher-Level Framework to AGTP Canonical Mapping

A wire example showing EXECUTE carrying an MCP tool invocation is provided in Appendix D. Additional examples for A2A and ACP are in the appendix-level composition section (Appendix G) and in [AGTP-COMPOSITION].

8.5. HTTP Gateway Sidecar

Operators that need to accept inbound HTTP/REST traffic on the same agent surface that serves AGTP *MAY* deploy an HTTP gateway sidecar alongside the AGTP daemon. The gateway is a parallel listener that accepts HTTP requests, translates them into AGTP method invocations, and dispatches them through the same response-finalization path as native AGTP traffic. The gateway is not part of the AGTP wire protocol; it is a deployment-time adoption ramp specified here so the translation contract is interoperable across implementations.

Translation rules:

1. *HTTP method → AGTP method.* The HTTP method is run through the server's policies.methods.aliases map per [AGTP-API]; the resolved canonical AGTP method is used for dispatch. The default alias seed (GET → FETCH, POST → CREATE, PUT → REPLACE, DELETE → REMOVE, PATCH → MODIFY) handles the five common verbs. The original HTTP method is preserved on the Attribution- Record as requested_method per [AGTP-IDENTIFIERS].

2. *HTTP path → AGTP path.* The path is forwarded verbatim, subject to the path grammar in [AGTP-API].

3. *HTTP headers → AGTP headers.* The gateway *MAY* forward selected headers (e.g., Authorization mapped to AGTP authentication context); the specific header forwarding policy is operator-defined.

4. *Response.* The AGTP response (status, headers, body) is translated into an HTTP response. The gateway *MUST* invoke the standard response-finalization path so the AGTP response carries a valid Attribution-Record; the resulting Audit-ID and Server-ID participate in the same per-agent chain as native AGTP responses.

Constraints:

* *The gateway **MUST* strip the Allow-RCNS header from inbound HTTP requests before dispatch.** HTTP callers cannot opt into RCNS through the gateway by design; runtime contract negotiation is a substrate feature for AGTP-native callers whose identity, trust tier, and scope are bound at the AGTP layer. Operators that want to expose RCNS to HTTP callers *MUST* layer AGTP-native authentication on top of the gateway, in which case the gateway is no longer the right abstraction.

* *The gateway is a translation surface, not a transport protocol.* Two AGTP servers exchanging traffic with one another *MUST NOT* use the gateway as their hop; AGTP is its own transport, and gateway-mediated AGTP-to-AGTP flows would lose the wire-level identity, attribution, and trust-posture guarantees this specification defines.

The reference implementation pattern is an operational module distributed as mod_http_gateway or equivalent, listening on a separate port from the AGTP listener (default 127.0.0.1:8080 for the gateway, port 4480 for AGTP), so the two protocols never share a socket.

8.6. Composition with External Identity Providers

AGTP identity (Agent-ID, Agent Certificate, Authority-Scope) and external identity-provider credentials (OAuth bearer tokens, OIDC id_tokens, SPIFFE SVIDs, enterprise IdP session tokens) answer different questions and *MUST* be treated as orthogonal axes:

* *AGTP identity answers "which agent is making this call."* Hash-anchored on the Agent Genesis (Section 6.5.2), presented via the Agent-ID header, optionally bound to a client certificate per [AGTP-CERT], scoped by Authority-Scope. The trust root is the Genesis-issuer registrar.

* *External IdP credentials answer "on whose behalf is the agent acting."* A human principal, a service account, or a workload identity in an enterprise identity stack. The trust root is the IdP.

The two axes compose. AGTP places no constraints on how external credentials are validated; servers *MAY* require external credentials on specific methods, *MAY* lift a named claim from the validated credential into the Attribution-Record, and *MUST* treat the failure modes of the two axes as independent.

8.6.1. Three Composition Patterns

Pattern 1: AGTP identity only. Agent-ID plus optional Agent Certificate plus Authority-Scope. No external IdP. Trust anchored in the Genesis-issuer registrar. Appropriate for closed agent ecosystems where the registrar is the trust root and no human-on-whose-behalf claim is required.

Pattern 2: AGTP identity plus external IdP credential. The request carries Agent-ID (wire-layer identity) and an Authorization header carrying an IdP-issued credential (application-layer principal). Both ride on the same request. AGTP Authority-Scope and external authorization scope (e.g., OAuth scopes) are independent: Authority-Scope is the agent's _capacity_ to act, the IdP scope is the _delegation_ from a principal authorizing the agent to do so on its behalf. A server's [policies.oauth] block (defined operationally outside this document) declares which methods require an external credential and which validator processes the credential. On validation success, the configured claim (typically sub) is lifted into the request context as the acting-principal identifier and stamped on the Attribution-Record per [AGTP-IDENTIFIERS].

Pattern 3: OIDC-federated Genesis-issuer trust. The Genesis-issuer key itself is attested by an IdP. Trust anchors that would otherwise be a static trusted-registrars list are replaced by OIDC discovery: the verifier configures one or more OIDC anchors (discovery URL + trusted issuer), fetches the JWKS at runtime, and confirms the issuer_public_key on the Agent Genesis matches one of the IdP's published keys. The Genesis schema does not change; the change is the resolution path the verifier uses to decide whether a recorded issuer key is trusted. See [AGTP-TRUST] for the trust-anchor schema.

8.6.2. Authorization Header

AGTP servers process the Authorization request header per HTTP semantics ([RFC9110]). The header value is opaque to AGTP itself; the configured validator interprets it. Common forms include:

* Bearer TOKEN — OAuth 2.0 / OIDC bearer tokens.

* Other schemes (Basic, Digest, custom schemes) — passed through to the configured validator unchanged.

Validation failures *MUST* return 401 Unauthorized with a response body carrying a structured reason from the vocabulary:

Reason	Meaning
oauth-required	The invoked method is in the server's required_on_methods list and no Authorization header was presented
oauth-invalid	An Authorization header was presented and the configured validator rejected it (bad signature, expired token, untrusted issuer, etc.)

Table 39: OAuth Composition 401 Reason Vocabulary

The 401 codes for external-credential failures are the same 401 code AGTP uses for other authentication failures; the reason field disambiguates.

8.6.3. Token Opacity and Attribution

AGTP *MUST NOT* stamp the raw Authorization header value or the raw token onto the Attribution-Record. The Attribution-Record is signed and may be replayed by chain walkers and audit consumers; embedding bearer tokens would create a credential-disclosure surface. Only the validated, lifted claim (the acting_principal_id) appears on the Attribution-Record per [AGTP-IDENTIFIERS].

8.6.4. Backward Compatibility

The OAuth composition surface is opt-in. Servers with no [policies.oauth] configuration process requests identically to servers built before this section was written. Requests without an Authorization header on servers that do not require one are dispatched normally. This preserves Pattern 1 deployments byte-for-byte.

9. Merchant Identity and Agentic Commerce Binding

9.1. Overview

AGTP specifies agent-side identity through the Agent Birth Certificate, canonical Agent-ID, Agent Identity Document, and Trust Tier model defined in Section 5. PURCHASE invocations carrying payments:purchase in the Authority-Scope header and a Budget-Limit constraint are fully governed on the sending side. The receiving side of a PURCHASE -- the merchant counterparty -- does not have an equivalent protocol-level identity in the base specification.

Version 04 of AGTP introduces normative integration hooks for the AGTP Merchant Identity and Agentic Commerce Binding specification [AGTP-MERCHANT], which defines the merchant-side identity model. The integration is hook-based: this document registers the required headers, status code, and Authority-Scope domains; the detailed semantics, Merchant Manifest Document schema, Merchant Birth Certificate structure, and counterparty verification procedure are specified in the companion.

9.2. Merchant Identity Headers (Summary)

PURCHASE invocations in a fully conformant v04 deployment carry the following additional headers:

* Merchant-ID: canonical identifier of the intended merchant counterparty.

* Merchant-Manifest-Fingerprint: SHA-256 fingerprint of the Merchant Manifest Document the requesting agent verified during pre-flight counterparty verification.

* Intent-Assertion: detached JWT carrying principal-authorized purchase intent, forwardable to payment networks as standalone evidence.

* Cart-Digest: digest of a structured cart returned by a prior QUOTE invocation, binding this PURCHASE to that cart.

Full field definitions, wire examples, and security requirements are in [AGTP-MERCHANT].

9.3. 458 Counterparty Unverified (Summary)

Receiving servers *MUST* return 458 Counterparty Unverified on PURCHASE invocations that fail merchant identity verification: missing Merchant-ID or Merchant-Manifest-Fingerprint headers, fingerprint mismatch, Merchant-ID mismatch, or a target merchant in any lifecycle state other than Active. 455 is a governance signal, parallel in role to 455 Scope Violation and 457 Zone Violation: *MUST* be logged; *MUST NOT* be retried without re-running counterparty verification.

9.4. Integration with PURCHASE, DISCOVER, and Attribution-Record

Three existing AGTP primitives interact with merchant identity:

* *PURCHASE*: Counterparty verification runs before PURCHASE is sent on the wire. A verified PURCHASE produces an Attribution- Record naming both the agent and the merchant cryptographically.

* *DISCOVER*: The DISCOVER method defined in [AGTP-DISCOVER] is extended by [AGTP-MERCHANT] to return Merchant Manifest Documents when the query carries result_type: "merchant", and to return mixed agent/merchant result sets when result_type: "any". The existing DISCOVER signature model, ranking model, and governance-zone enforcement apply unchanged.

* *Attribution-Record*: The Attribution-Record returned on PURCHASE includes merchant_id, merchant_fingerprint, and intent_assertion_jti fields when merchant identity binding is in effect. This produces a dual-party cryptographic record consumable by downstream audit and dispute-resolution processes without requiring those processes to speak AGTP.

9.5. Relationship to Payment Networks

The merchant identity model defined in this document is payment-rail neutral. It does not define payment credential handling, tokenized card-on-file representations, authorization messaging to card networks, or settlement. Payment networks wishing to extend protection, fraud coverage, or dispute handling to agent-initiated transactions consume the Intent-Assertion JWT and the Attribution-Record as verifiable inputs to their own authorization and dispute flows; no AGTP-layer integration is required on the payment-network side. The specific mapping between AGTP merchant identity artifacts and payment-network message formats is expected to be defined bilaterally between governance platforms and individual networks and is out of scope for this document.

10. Security Considerations

This section satisfies the mandatory IETF Security Considerations requirement. All AGTP implementations *MUST* address the considerations described here.

10.1. Mandatory TLS

All AGTP connections *MUST* use TLS 1.3 or higher. Implementations *MUST* reject connections using TLS 1.2 or below. Certificate validation follows standard PKI practices per [RFC5280]. Servers *MUST* present a valid certificate.

10.2. Agent Identity Verification: Three Levels

AGTP defines three distinct levels at which agent identity and Authority-Scope can be verified. Each level serves a different deployment profile and operational tradeoff. Understanding the distinction is essential for implementers: the AGTP Agent Certificate extension ([AGTP-CERT]) is OPTIONAL, and base AGTP provides cryptographic verification at the application layer without it.

Level 1 - Self-asserted headers (raw request fields). Every AGTP request *MUST* include the Agent-ID header, which references the agent identity document carrying the principal identifier and Authority-Scope. As a raw header value on an individual request, Agent-ID is self-asserted: a client writes the value into the request and the server records what was written. Level 1 verification is limited to mandatory logging and anomaly detection against the recorded stream. This is the minimum baseline every AGTP implementation provides.

Level 2 - Application-layer cryptographic verification (signed Agent Identity Document). A canonical Agent-ID resolves to a signed Agent Identity Document (Section 5.5) that carries the Birth Certificate's Authority-Scope grant and is signed by the governance platform that issued it. A verifier (including a stranger with no prior relationship to the agent's organization) can cryptographically verify identity and scope at the application layer by performing the following steps:

1. Resolve the canonical Agent-ID to retrieve the signed Agent Manifest Document.

2. Verify the governance platform's signature on the manifest against the platform's published key.

3. Confirm that the canonical Agent-ID in the manifest matches the hash of the Agent Genesis.

4. Read the Authority-Scope grant from the verified manifest.

Level 2 verification is available in base AGTP without the Agent Certificate extension. It is the identity mechanism the protocol depends on. Self-asserted headers (Level 1) are bound to verified identity (Level 2) by the resolver's retrieval of the signed manifest for the declared canonical Agent-ID.

Level 3 - Transport-layer cryptographic verification (AGTP-CERT extension). The AGTP Agent Certificate extension [AGTP-CERT] binds the canonical Agent-ID and Authority-Scope to an X.509 v3 certificate

presented during TLS 1.3 mutual authentication. The principal identifier remains in the agent identity document referenced by Agent-ID. Level 3 accelerates the Level 2 check to the TLS handshake and enables Scope-Enforcement Points (SEPs) to verify Authority-Scope at O(1) per-request cost without application-layer access. Level 3 is an acceleration and enforcement path for Level 2, not a replacement of it. Deployments that require line-rate scope enforcement at infrastructure layers (load balancers, governance gateways) *SHOULD* implement [AGTP-CERT].

> Note: The Agent Certificate extension and the Agent Genesis mechanism may be subject to pending intellectual property claims. See Section 7.7 and the IPR Notice preceding the Abstract for details. The licensor is prepared to grant a royalty-free license to implementers.

Every AGTP server *MUST* log the Agent-ID value from every request and the principal identifier resolved from the corresponding agent identity document, creating an attributable audit trail at Level 1 even in deployments that do not implement Level 2 retrieval or Level 3 transport binding.

10.3. Authority Scope Enforcement

The Authority-Scope header declares what actions the agent is authorized to take. Compliant AGTP servers *MUST* parse the Authority-Scope on every request, return 455 Scope Violation for any method that exceeds declared scope, and log all scope violations for audit purposes. At Level 1, scope declarations are self-asserted in the request header, analogous to scope assertions in OAuth 2.0 [RFC6749]. At Level 2, scope is cryptographically verifiable through the signed Agent Identity Document; servers *SHOULD* retrieve and verify the manifest for any Agent-ID whose declared scope exceeds read-only operations. Level 3 cryptographically signed and infrastructure-enforced scopes are defined in [AGTP-CERT].

10.4. Threat Model

10.4.1. Agent Spoofing

Threat: A malicious actor forges the Agent-ID header to impersonate a trusted agent. Mitigation: Level 2 application-layer verification binds a declared Agent-ID to the signed Agent Identity Document retrieved via canonical ID resolution. A forged Agent-ID either fails to resolve or resolves to a manifest whose signature cannot be verified against the claimed governance platform’s published key. Level 3 raises the mitigation to the TLS handshake via [AGTP-CERT]. Implementations *SHOULD* retrieve and verify the manifest for any

Agent-ID carrying scope beyond read-only query operations. Mandatory Level 1 logging provides an anomaly-detection baseline for deployments that do not perform active verification on every request.

10.4.2. Authority Laundering

Threat: An agent claims an Authority-Scope broader than what it was granted. Mitigation: server-side scope enforcement; 262 Authorization Required (body type scope-required) returned and logged. In DELEGATE chains, each hop's scope *MUST* be a strict subset of the delegating agent's scope.

10.4.3. Delegation Chain Poisoning

Threat: A malicious agent inserts itself into a delegation chain. Mitigation: each hop of a DELEGATE chain *MUST* be logged with the delegating agent's Agent-ID, the sub-agent's Agent-ID, and the declared Authority-Scope. Servers processing a delegated request *MUST* be able to reconstruct the delegation sequence from log data, sub-agent identity documents, and Authority-Scope subset verification. 551 Authority Chain Broken is returned when any link in the sequence is unverifiable. Full mitigation requires [AGTP-CERT] for signed delegation tokens.

10.4.4. Denial of Service via High-Frequency Agent Traffic

Threat: Agents that are compromised, misconfigured, or adversarial generate extremely high request volumes. Mitigation: 429 Rate Limited status code. Rate limiting *SHOULD* be applied per Agent-ID and per resolved principal identifier (obtained from the agent identity document referenced by Agent-ID). When [AGTP-CERT] is deployed, per-Agent-ID quotas can be cryptographically tied to verified identity, preventing quota evasion through Agent-ID spoofing.

10.4.5. Session Hijacking

Threat: An attacker intercepts or forges a Session-ID. Mitigation: mandatory TLS protects sessions in transit. Session-IDs *MUST* be cryptographically random with minimum 128 bits of entropy. Servers *MUST* validate that Session-ID, Agent-ID, and TLS client identity are consistent.

10.4.6. Escalation Suppression

Threat: A compromised agent or intermediary suppresses ESCALATE requests, preventing human oversight. Mitigation: compliant implementations *MUST* route ESCALATE requests directly to the declared escalation handler without modification. Intermediaries *MUST NOT* drop, delay, or modify ESCALATE requests. Escalation handlers *SHOULD* implement independent receipt confirmation.

10.4.7. Agent Genesis Spoofing

Threat: A malicious actor fabricates an Agent Genesis to claim a legitimate agent's identity or construct a false identity with elevated trust. Mitigation: Agent Genesis documents are issued only by governance platforms that have completed one of the three Tier 1 verification paths (Section 5.2). For dns-anchored registrations, the governance platform *MUST* verify DNS ownership of the claimed org_domain before issuance. For log-anchored registrations, the governance platform *MUST* submit the Agent Genesis to a transparency log per [RFC9162] / [RFC9943] and record the inclusion proof in the registry; tampering with a log-anchored Agent Genesis is detectable by any party with log access. For hybrid registrations, both DNS and blockchain address ownership are verified. In the base spec, mandatory logging provides auditability. Full mitigation requires [AGTP-CERT] for cryptographically bound Agent Genesis verification at the transport layer. Governance platforms *MUST* treat any ACTIVATE request whose computed canonical Agent-ID matches an existing registry record as a collision attack and *MUST* reject it.

10.4.8. Domain Transfer Identity Hijacking

Threat: An attacker acquires an expired domain to inherit the agent registry and trust history of prior registrants. Mitigation applies to dns-anchored and hybrid Tier 1 agents: agents under an expired domain are automatically Suspended within 24 hours of domain expiry detection. A new owner of the domain *MUST NOT* inherit prior agent registrations. See Section 9.6 for the full domain expiry policy. log-anchored Tier 1 agents are unaffected by this threat because their verification evidence is the transparency log inclusion proof rather than DNS ownership.

10.4.9. Attribution Forgery

Threat: A malicious agent submits a fabricated or replayed Attribution-Record to claim credit for an action it did not perform, or to conceal the true execution context of an action it did perform.

Mitigation: Attribution-Records used for accountability or admission to an audit trail *MUST* be signed with the agent's governance key. The signature *MUST* cover the full record including the Task-ID, Agent-ID, method, timestamp, and result hash. When [AGTP-CERT] is deployed, the signature is verified at the transport layer against the agent's X.509 certificate. For high-stakes domains, RATS attestation evidence in the Attribution-Record per [RFC9334] provides hardware-rooted proof of execution context that cannot be forged without compromising the attesting environment itself. Attribution-Record signatures *MUST* be verified before the record is admitted to an audit trail. Unverified records *MUST* be logged with a signature_unverified flag and *MUST NOT* be treated as authoritative for compliance purposes.

The alg: none fallback permitted on the Attribution-Record response header (Section 5.5.4) exists to preserve wire format and Audit-ID chain construction in development and transition deployments that have not yet provisioned a manifest signing key. An alg: none record carries no anti-forgery claim. Consumers *MUST* treat alg: none records as equivalent to unverified records for the purposes of this mitigation: they *MUST NOT* be admitted to an audit trail as authoritative, and they *MUST NOT* satisfy any accountability requirement that depends on cryptographic signing. Production deployments *MUST* configure a manifest signing key.

10.5. Privacy Considerations

Agent identity carried on requests, and the agent identity document referenced by Agent-ID, carry information about agent behavior that may be sensitive:

* Agent-ID together with the resolved principal identifier may reveal organizational structure

* Session-ID and Task-ID reveal workflow patterns

* Delegation-Chain (reserved for future revisions) would reveal multi-agent architecture

AGTP logs containing these fields *MUST* be treated as sensitive operational data. Operators *MUST* implement appropriate access controls, retention limits, and data minimization practices consistent with applicable privacy regulations.

Where privacy-preserving attribution is required, implementations *MAY* use pseudonymous Agent-IDs with a separate trusted resolution service. The architecture for pseudonymous agent identity resolution is reserved for a future companion document.

10.6. Denial-of-Service Considerations

AGTP's agent identity provides a mechanism for more precise denial-of-service mitigation than is possible with HTTP. Rate limiting *SHOULD* be applied per Agent-ID and per resolved principal identifier (obtained from the agent identity document) in addition to per-IP-address controls.

When [AGTP-CERT] is deployed, per-Agent-ID rate limiting can be cryptographically tied to verified agent identity, preventing quota evasion through Agent-ID rotation. Implementations planning high-volume governed agent deployments *SHOULD* plan for [AGTP-CERT] as part of their denial-of-service mitigation strategy.

Additional recommended mitigations: traffic-shaping by request class once the Priority header is normatively specified (reserved for v01+ per Section 5.5); and circuit breaker patterns for ESCALATE request floods.

10.7. Intellectual Property Considerations

The core AGTP specification, including all base methods, header fields, status codes, connection model, and IANA registrations defined in this document, is intended for open implementation without royalty obligation.

Certain elements referenced in this document may be subject to pending patent applications by the author, specifically:

* The Agent Certificate extension [AGTP-CERT], which provides cryptographic binding of agent identity and authority scope to AGTP header fields.

* The ACTIVATE method, which provides AGTP-native transmission and activation of governed agent packages.

* The Agent Genesis mechanism (Section 5.7), which provides the genesis identity record and canonical Agent-ID derivation process for AGTP-registered agents.

* The .agent file format specification, an open packaging format for AI agents.

* The .nomo file format specification, a governed packaging format for AI agents with cryptographic governance binding.

Implementers of the core AGTP specification are not affected by any intellectual property claims on these extensions and associated formats.

The licensor is prepared to grant a royalty-free license to implementers for any patent claims that cover contributions in this document and its referenced extensions, consistent with the IETF's IPR framework under [RFC8179].

IPR disclosures have been filed with the IETF Secretariat and are available at: https://datatracker.ietf.org/ipr/

11. IANA Considerations

This document records the following IANA registrations.

11.1. Port Assignment

The following service names and port number are registered in the IANA Service Name and Transport Protocol Port Number Registry:

Service Name	Port	Transport	Description
agtp	4480	TCP	Agent Transfer Protocol over TCP/TLS
agtp-quic	4480	UDP	Agent Transfer Protocol over QUIC

Table 40: AGTP Port Assignments

The unified port assignment (4480 for both TCP and UDP under a single conceptual agtp service) follows the precedent set by HTTPS (443/TCP and 443/UDP for HTTP/3). The transport is distinguished at the protocol level rather than at the port number.

Contact: Chris Hood, chris@nomotic.ai

Reference: This document

11.2. AGTP Method Registry

Establishment of an IANA registry: Agent Transfer Protocol Methods.

Registry name: Agent Transfer Protocol Methods

Registration procedure: Expert Review per [RFC8126], with the additional requirement that each registration be accompanied by a published specification, at minimum a publicly available Internet-Draft or equivalent document. The Designated Expert *SHOULD* verify that the proposed method name is unique, the reference specification is publicly accessible, the method definition includes the required fields (parameters, response structure, idempotency, error codes, security considerations), and the method conforms to the AGTP-API specification [AGTP-API].

Reference: This document

Initial registrations (the eighteen-method protocol floor):

Method	Group	Status	Reference
QUERY	Cognitive	Permanent	Section 7.2
DISCOVER	Cognitive	Permanent	Section 7.2
DESCRIBE	Cognitive	Permanent	Section 7.2
INSPECT	Cognitive	Permanent	Section 7.2.4
SUMMARIZE	Cognitive	Permanent	Section 7.2
PLAN	Cognitive	Permanent	Section 7.2
PROPOSE	Cognitive	Permanent	Section 7.2
EXECUTE	Mechanics	Permanent	Section 7.2
DELEGATE	Mechanics	Permanent	Section 7.2
ESCALATE	Mechanics	Permanent	Section 7.2
CONFIRM	Mechanics	Permanent	Section 7.2
SUSPEND	Mechanics	Permanent	Section 7.2
NOTIFY	Mechanics	Permanent	Section 7.2
ACTIVATE	Lifecycle	Permanent	Section 7.2.14
DEACTIVATE	Lifecycle	Permanent	Section 7.2.15
REINSTATE	Lifecycle	Permanent	Section 7.2.17
REVOKE	Lifecycle	Permanent	Section 7.2.16
DEPRECATE	Lifecycle	Permanent	Section 7.2.18

Table 41: Initial AGTP Method Registry Entries (Eighteen-Method Floor)

The methods BOOK, SCHEDULE, LEARN, and COLLABORATE, present in the v06 core set, are demoted in v07 to Tier 2 standard extended methods and are registered through the AGTP-API method catalog ([AGTP-API]) rather than through this document.

11.3. AGTP Status Code Registry

Establishment of an IANA registry: Agent Transfer Protocol Status Codes.

Registry name: Agent Transfer Protocol Status Codes

Registration procedure: Expert Review + published specification required.

AGTP-specific status code numbers are deliberately chosen from ranges unassigned in the IANA HTTP Status Code Registry to avoid semantic collision with HTTP status codes that may appear in payloads carried by AGTP method invocations.

Two status codes used by AGTP retain their HTTP code numbers (408 and 410) but carry AGTP-specific semantics. They are registered here with text describing the AGTP semantic.

The following AGTP status codes are registered with full definitions:

Code	Name	Definition	Reference
261	Negotiation In Progress	The service has received a PROPOSE request and is evaluating it asynchronously. The response body *MUST* include a proposal_id and an estimated evaluation duration. The agent retrieves the terminal status by invoking QUERY /proposals/{proposal_id} until a 263 (Proposal Approved) or 463 (Proposal Rejected) response is returned. Server policy controls whether async evaluation is offered.	[AGTP-API]
262	Authorization Required	The request requires credential establishment, additional authorization scope, or consent that is not yet present. The response body *MUST* specify which authorization condition	[AGTP-API]

		applies: scope-required (the endpoint requires Authority-Scope the agent has not declared), wildcards-required (the request is an ad-hoc method invocation and the wildcards consent on either the agent identity document or the server policy is absent), credentials-missing (the server requires credentials such as AGTP-CERT or OAuth-scoped token before evaluating the request), or anonymous-discovery-disabled (the server requires authenticated identity for manifest retrieval and the request is unauthenticated). Returned for PROPOSE-time authorization, endpoint-dispatch scope checks, ad-hoc invocation wildcards refusal, and discovery requests blocked by policy.	
263	Proposal Approved	The service has accepted the PROPOSE request and instantiated the proposed endpoint. The response body *MUST* contain a complete AGTP-API endpoint definition for the instantiated endpoint, a synthesis_id identifying the synthesized endpoint for follow-on invocation, and expires_at indicating when the synthesis expires. *MAY* include a persistent boolean and granted_duration indicating the actual duration the server granted (which may be less than the agent requested, bounded by server policy).	[AGTP-API]
405	Method Not	The method is recognized and	[AGTP-API]

	Allowed	the path is valid, but the server's policy or registry does not expose this combination. The response body *MUST* list allowed methods for the path and any redirects from the manifest's policies.methods sub-block. The agent *MAY* PROPOSE the combination if it is not exposed by policy.	
408	Timeout	The method's declared TTL expired before execution completed. AGTP-specific semantics distinct from HTTP's request-timeout: applies to AGTP method TTL rather than transport request timeout. *MUST* be logged.	Section 5.6
410	Gone	The Agent-ID is permanently retired through REVOKE of the underlying Agent Genesis. AGTP-specific semantics distinct from HTTP's resource-removed: applies to permanent identity retirement. A Deprecated agent does *NOT* return 410 and continues to serve traffic; deprecation signals planned end-of-life rather than retirement. The canonical Agent-ID of a retired agent *MUST NOT* be retried.	Section 5.6
455	Scope Violation	The requested action is outside a declared scope dimension other than Authority-Scope, rate-limit, budget, or zone (which have dedicated codes 262, 429, 456, 457 respectively). Typical applications include token-based scope violations and query-based scope	Section 5.6

		violations where the operator defines a scope dimension outside the standard set. The server *MUST* log this event. The agent *MUST NOT* retry the same request without modifying its scope declaration. Governance signal, not a protocol error.	
456	Budget Exceeded	The requested method execution would exceed the resource limits declared in the Budget-Limit request header. The agent *MUST NOT* retry without modifying the Budget-Limit or reducing request scope. Governance signal; *MUST* be logged.	Section 5.6
457	Zone Violation	The request would route outside the network boundary declared in the AGTP-Zone-ID header. SEP-enforced. The agent *MUST NOT* retry without modifying the AGTP-Zone-ID or obtaining explicit cross-zone authorization. *MUST* be logged.	Section 5.6
458	Counterparty Unverified	The merchant counterparty in a PURCHASE invocation failed identity verification. Returned when the Merchant-ID or Merchant-Manifest-Fingerprint request headers are absent, when the fingerprint does not match the receiving server’s current Merchant Manifest Document, when the Merchant-ID does not match the server’s canonical ID, or when the merchant is in a non-Active lifecycle state. Governance signal; *MUST* be logged. Full definition in [AGTP-MERCHANT].	[AGTP-MERCHANT]

459	Method Violation	The method name is not in the AGTP-API approved method catalog. The method itself is the problem. The response body *MUST* identify the unrecognized method and *SHOULD* reference the AGTP-API method catalog version in effect. The agent *MUST* select a different method before retrying, or *MAY* PROPOSE the method if no suitable replacement exists.	[AGTP-API]
460	Endpoint Violation	The endpoint path violates AGTP-API path grammar. A path segment matches an approved method name (case-insensitive), indicating method-name leakage into the path. The response body *MUST* identify the offending path segment. The agent *MUST* restructure the path before retrying.	[AGTP-API]
461	RCNS Contract Available	The requested endpoint is not registered but the server is prepared to synthesize a contract for it. The response body *MUST* carry a contract preview including the resolved method, path, and synthesis_id; the caller *MAY* accept the contract by re-issuing the request with the Contract-Synthesized header or decline by ignoring the response. Returned in the confirm-first RCNS delivery mode per [AGTP-API].	[AGTP-API]
462	Reserved	Reserved for AGTP expansion.	Section 5.6
463	Proposal Rejected	The service cannot or will not instantiate the proposed endpoint. Returned in	[AGTP-API]

		response to PROPOSE. The response body *MUST* include a structured reason field with one of the values out-of-scope, policy-refused, composition-impossible, ambiguous, or synthesis-disabled, an explanation string, and *MAY* include an optional counter_proposal suggesting a related endpoint the server would accept.		
464	RCNS No Contract	An RCNS synthesis attempt was made but no contract could be delivered. The response body *MUST* include a structured reason field with one of the values rcns-disabled, trust-tier-insufficient, composition-impossible, synthesis-error, contract-not-yours, or contract-revoked, and an explanation string. See [AGTP-API].	[AGTP-API]	
465	Reserved	Reserved for AGTP expansion.	Section 5.6	
550	Delegation Failure	A sub-agent to which a task was delegated via the DELEGATE method failed to complete the task within the declared deadline or returned an error. The response body *SHOULD* contain the sub-agent’s error details.	Section 5.6	
551	Authority Chain Broken	One or more entries in the delegation sequence cannot be verified as part of a valid and continuous delegation chain. The specific unverifiable entry *SHOULD* be identified in the response body. The server *MUST* log this event.	Section 5.6	
552	Reserved	Reserved for AGTP expansion.	Section 5.6	

553	Reserved	Reserved for AGTP expansion.	Section 5.6
554	Reserved	Reserved for AGTP expansion.	Section 5.6
555	Reserved	Reserved for AGTP expansion.	Section 5.6

Table 42: AGTP-Specific Status Code Definitions

11.4. Media Type Registry

This section is the master registry of all AGTP-family media types. Types defined by companion specifications are listed here for cross-document discoverability; their normative definitions remain in the defining document.

Media Type	Use	Defining Reference	IANA Status
application/ vnd.agtp+json	AGTP method request/ response bodies (JSON)	Section 5.7	Planned (this document)
application/ vnd.agtp+yaml	AGTP method request/ response bodies (YAML)	Section 5.7	Planned (this document)
application/ vnd.agtp.identity+json	Agent Identity Document (JSON)	Section 6.3	Vendor-tree registration submitted
application/ vnd.agtp.identity+yaml	Agent Identity Document (YAML)	Section 6.3	Vendor-tree registration submitted
application/ vnd.agtp.manifest+json	AGTP server manifest	[AGTP-API]	Planned (AGTP-API)
application/ vnd.agtp.endpoint+json	AGTP-API endpoint definition	[AGTP-API]	Planned (AGTP-API)

Table 43: AGTP Media Type Registrations

"Vendor-tree registration submitted" indicates that the registration application has been filed with IANA against the application/vnd.* vendor tree and is awaiting processing. "Planned (this document)" and "Planned (AGTP-API)" indicate types for which registration applications will be filed concurrent with publication of the defining document. Standards-tree promotion of the full set is anticipated at RFC publication of the AGTP family.

11.5. Header Field Registry

AGTP header fields are distinct from HTTP header fields and are registered in a new IANA registry: Agent Transfer Protocol Header Fields.

Registry name: Agent Transfer Protocol Header Fields

Registration procedure: Expert Review + published specification required.

AGTP does not reuse the HTTP Field Name Registry, as AGTP header fields have different semantics, applicability, and versioning constraints from HTTP fields. HTTP header fields are not automatically valid in AGTP, and AGTP header fields are not valid HTTP fields.

Initial registrations (all Permanent): Agent-ID, Authority-Scope, Session-ID, Task-ID, Delegation-Chain, Server-ID, Attribution-Record, Continuation-Token, Supported-Methods, Cost-Estimate, Attestation-Evidence, Merchant-ID, Merchant-Manifest-Fingerprint, Intent-Assertion, Cart-Digest, AGTP-API-Version, AGTP-Catalog-Warning, AGTP-Endpoint-Warning.

The four merchant-related headers are defined in [AGTP-MERCHANT] and registered concurrently with this document. The three API-related headers (AGTP-API-Version, AGTP-Catalog-Warning, and AGTP-Endpoint-Warning) are defined in [AGTP-API] and registered concurrently.

Headers reserved for future revisions (Priority, TTL, Budget-Limit, AGTP-Zone-ID, Content-Schema, Telemetry-Export) are not registered in this revision. They will be registered alongside the future revision that specifies their normative semantics.

11.6. URI Scheme Registration

The agtp:// URI scheme is registered per [RFC7595]. Full registration template is documented in Section 5.1.8 of this document.

11.7. AGTP Budget Unit Registry

Establishment of a new IANA sub-registry: Agent Transfer Protocol Budget Units.

Registry name: Agent Transfer Protocol Budget Units

Registration procedure: Expert Review per [RFC8126]. New unit

registrations *MUST* define: unit name (lowercase ASCII, no spaces or special characters), semantic description, value format (integer or decimal), whether fractional values are permitted, and a reference specification. Units representing financial denominations *MUST* specify the currency and *MUST* define precision (decimal places). The Designated Expert *SHOULD* verify that the proposed unit does not duplicate an existing registration and that the value format is unambiguous.

Reference: This document

Initial registrations:

Unit	Description	Value Format	Fractional
tokens	Language model token consumption	Integer	No
compute-seconds	CPU/GPU compute time in seconds	Decimal	Yes
USD	US Dollar financial limit	Decimal, 2 places	Yes
EUR	Euro financial limit	Decimal, 2 places	Yes
GBP	Pound Sterling financial limit	Decimal, 2 places	Yes
calls	Downstream API call count	Integer	No

Table 44: Initial AGTP Budget Unit Registry Entries

11.8. Agent Registry Retention Policy

The AGTP registry *MUST* retain records for all registered agents regardless of lifecycle state. The following minimum retention periods apply:

Lifecycle State	Minimum Retention Period
Active	Duration of Active state + 7 years
Suspended	Duration of Suspended state + 7 years
Revoked	10 years from revocation date
Deprecated	7 years from deprecation date

Table 45: AGTP Registry Minimum Retention Periods

The 7-year minimum reflects common enterprise compliance requirements (SOX, GDPR audit trails, HIPAA). Governance platform operators in regulated industries *SHOULD* extend these minimums to match applicable regulatory requirements.

The retained record for a Revoked or Deprecated agent *MUST* include:

* Canonical Agent-ID (permanently retired, not reissued)
* Agent label and org domain at time of registration
* Trust tier at time of registration
* Activation date and activating principal
* Revocation or deprecation date, initiating principal, and reason code
* Genesis audit record hash (pointer to immutable audit log)
* Full Agent Genesis (archived, not publicly accessible)
* All lifecycle state transitions with timestamps

The retained record *MUST NOT* contain package executable contents, active session data, or Authority-Scope grant tokens.

11.8.1. Domain Name Expiry Interaction

If an organization’s org_domain expires or transfers to a new owner:

1. All Active agents registered under the expired domain *MUST* be automatically Suspended within 24 hours of domain expiry detection.

2. The governance platform *MUST* notify the registered principal contact before suspension takes effect, with a minimum notice period of 30 days if domain expiry was predictable.

3. Suspended agents under an expired domain transition to Deprecated state after 90 days if the domain has not been renewed.

4. A new owner of the domain *MUST NOT* inherit prior agent registrations. New ACTIVATE transactions are required.

This policy prevents domain-transfer-based identity hijacking in which an attacker acquires an expired domain to claim the trust history of agents that operated under it.

12. References

12.1. Normative References

[AGTP-API] Hood, C., "AGTP-API: Verbs, Paths, Endpoints, and Synthesis", Work in Progress, Internet-Draft, draft-hood-agtp-api-01, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-api-01>.

[AGTP-IDENTIFIERS] Hood, C., "AGTP Identifier Stack: Identifiers and Per-Agent Audit Chain", Work in Progress, Internet-Draft, draft-hood-agtp-identifiers-01, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-identifiers-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>.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/rfc/rfc6335>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[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>.

[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>.

[RFC8179] Bradner, S. and J. Contreras, "Intellectual Property Rights in IETF Technology", BCP 79, RFC 8179, DOI 10.17487/RFC8179, May 2017, <https://www.rfc-editor.org/rfc/rfc8179>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/rfc/rfc8555>.

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

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

12.2. Informative References

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

[ACP] IBM Research, "Agent Communication Protocol", 2025.

[AGTP-CERT] Hood, C., "AGTP Agent Certificate Extension", Work in Progress, Internet-Draft, draft-hood-agtp-agent-cert-01, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-agent-cert-01>.

[AGTP-COMPOSITION] Hood, C., "AGTP Composition with Agent Group Messaging Protocols", Work in Progress, Internet-Draft, draft-hood-agtp-composition-00, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-composition-00>.

[AGTP-DISCOVER] Hood, C., "AGTP Agent Discovery and Name Service", Work in Progress, Internet-Draft, draft-hood-agtp-discovery-00, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-discovery-00>.

[AGTP-LOG] Hood, C., "AGTP Transparency Log Protocol", Work in Progress, Internet-Draft, draft-hood-agtp-log-02, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-log-02>.

[AGTP-MERCHANT] Hood, C., "AGTP Merchant Identity and Agentic Commerce Binding", Work in Progress, Internet-Draft, draft-hood-agtp-merchant-identity-02, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-merchant-identity-02>.

[AGTP-TRUST] Hood, C., "AGTP Trust and Verification Specification", Work in Progress, Internet-Draft, draft-hood-agtp-trust-01, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-trust-01>.

[AGTP-WEB3] Hood, C., "AGTP Web3 Bridge Specification", Work in Progress, Internet-Draft, draft-hood-agtp-web3-bridge-00, 2026, <https://datatracker.ietf.org/doc/html/draft-hood-agtp-web3-bridge-00>.

[ANP] "Agent Network Protocol", 2025.

[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.

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

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

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/rfc/rfc7540>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/rfc/rfc7595>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

[RFC9943] "*** BROKEN REFERENCE ***".

Appendix A. Changes from v07

Version 08 is a drift-cleanup revision. The protocol surface is unchanged; clarifications align spec wording with deployed implementation behavior.

A.1. Substantive Changes

The following substantive changes were made:

1. *Agent-ID echoed on responses.* The Response Headers table (Section 5.5.4) now documents the Agent-ID response header. When a request carries an Agent-ID header, the server's response *SHOULD* echo that value to correlate the response with the requesting agent. This parallels the existing Task-ID echo behavior and reflects the contract that deployed implementations have provided since v07. Server-ID and Agent-ID remain semantically distinct on responses: Server-ID names the host that produced the response, Agent-ID (when echoed) names the agent the response is being returned to.

2. *Attribution-Record tightened to JWS Compact and always emitted.* The Attribution-Record row in Section 5.5.4 now specifies the JWS Compact serialization per [RFC7515] normatively, calls out the extended payload (identifier-chain fields and previous_audit_id) by reference to [AGTP-IDENTIFIERS], and changes the emission contract from "SHOULD when manifest signing is configured" to "MUST," with an explicit alg: none fallback when no signing key is configured. The fallback preserves wire format and Audit-ID chain construction; it carries no anti-forgery claim and is addressed in the Attribution Forgery security consideration.

3. *Audit-ID, Response-ID, Owner-ID, and Request-ID response headers added to the table.* These headers were stamped by deployed implementations since v07 and defined normatively in [AGTP-IDENTIFIERS]; the base draft's Response Headers table now lists them with their emission contracts. Audit-ID is required when Attribution-Record is present; Response-ID is required on every response; Owner-ID is SHOULD on every response; Request-ID echoes the request's value when present.

4. *Normative reference to RFC 7515 added.* Required by the tightened Attribution-Record description. Informative reference to [AGTP-IDENTIFIERS] (draft-hood-agtp-identifiers) added; the identifier stack and extended Attribution-Record schema are specified in that companion draft.

5. *Agent Genesis schema cleaned up.* The Agent Genesis Contents table (Section 6.5.2) drops the certificate_hash field that no longer exists under the locked taxonomy. The canonical Agent-ID is defined explicitly as sha256(canonical_form(Agent_Genesis_without_signature)); the agent_id field in the Genesis is populated for relying-party

convenience and verifiers *MUST* independently recompute the hash before trusting it. The signature field is described normatively as a detached Ed25519 signature over the canonical form (with the signature field excluded). The canonical-form requirement is stated explicitly: deterministic serialization, sorted keys, no insignificant whitespace, UTF-8, identical bytes across implementations. The Agent Genesis to AGTP Protocol Mapping table drops the certificate_hash row; the chain prose is rewritten to name the document-itself-hashes-itself model. References to certificate_hash in the ISSUE flow and in the collision-attack security consideration are updated to refer to the recomputed canonical Agent-ID. These changes align the spec with the [AGTP-IDENTIFIERS] definition of the canonical Agent-ID and with deployed implementations.

6. *Agent Identity Document trust-posture surfacing.* The owner_id field is added to the RECOMMENDED fields of the Agent Identity Document. The trust_tier, verification_path, and owner_id field descriptions now reference the trust-posture loading precedence rule defined in [AGTP-TRUST] (explicit declaration > Genesis-derived fallback > conservative default). The verification_path field description is corrected to include org-asserted as the Tier 2 value (earlier text listed only the three Tier 1 paths). The cross-reference to the Owner-ID response header (Section 5.5.4) is added. These changes document the wire surface and the loading contract that v07-conformant implementations have shipped.

7. *INSPECT promoted to the protocol-level method floor.* The protocol floor expands from twelve methods to thirteen with the addition of INSPECT, a cognitive method that retrieves stored Attribution-Records (by Audit-ID), per-agent chain heads, or lifecycle log entries from the responding server. INSPECT is specified in Section 7.2.4 and registered in the IANA AGTP Method Registry. INSPECT is the standardized lookup interface for Audit-IDs referenced in [AGTP-IDENTIFIERS]: a chain walker that holds an Audit-ID invokes INSPECT target=audit audit_id={hex} to retrieve the underlying JWS and walk the chain by previous_audit_id. The floor count is updated in the protocol-surface descriptions throughout the abstract, terminology, methods section, and IANA Method Registry table.

8. *role field added to Agent Identity Document.* A new RECOMMENDED role field is added to the Agent Identity Document with values agent (default) or merchant. Capability roles ride on the Agent Identity Document rather than the Agent Genesis: identity is permanent (hash-anchored on Genesis), capability is mutable (editable on the Identity Document between server restarts).

The merchant value is the normative trigger for PURCHASE counterparty verification per [AGTP-MERCHANT]; future revisions may define additional role values. Implementations encountering an unknown role value *MUST* treat the agent as the default agent role. AGTP-MERCHANT seriesinfo updated to v02, which retires the v01 Merchant Genesis as a separate document type in favor of this unified model.

9. *Lifecycle methods promoted to the protocol-level floor.* The protocol floor expands from thirteen methods to sixteen with the addition of three Lifecycle methods: ACTIVATE (Section 7.2.14), DEACTIVATE (Section 7.2.15), and REVOKE (Section 7.2.16). The floor is now organized as three groups (Cognitive, Mechanics, Lifecycle) rather than two; the cognitive / mechanics / lifecycle distinction is informational, not normative. The Method Category Taxonomy gains a sixth category, LIFECYCLE, alongside Acquire, Compute, Transact, Communicate, and Orchestrate. The stale paragraph describing ACTIVATE as an optional method extension in a separate companion is withdrawn; ACTIVATE, DEACTIVATE, and REVOKE are core methods that any conformant AGTP implementation *MUST* support. Each Lifecycle method produces a signed lifecycle event in the agent’s per-agent lifecycle stream and the event is retrievable via INSPECT target=lifecycle. The AGTP-LOG seriesinfo is updated to v02, which aligns the event-type triggering methods with the v08 Lifecycle group. Already-target-state transitions are no-ops with 200 OK and a noop: true flag; servers *MUST NOT* emit duplicate lifecycle events.

10. *Lifecycle group expanded with REINSTATE and DEPRECATE; Trust-Tier response headers added.* The protocol floor expands from sixteen to eighteen with the addition of two further Lifecycle methods: REINSTATE (Section 7.2.17), the explicit reversal path for DEACTIVATE, and DEPRECATE (Section 7.2.18), the controlled end-of-life signal that keeps the agent serving traffic while clients migrate. The Lifecycle group now contains five methods. Three new response headers are added to the Response Headers table: Trust- Tier, Verification-Path, and Trust-Warning, stamped on every response per the trust-posture loading rule in [AGTP-TRUST]. The headers allow relying parties to apply trust-tier-conditional policy on every exchange without consulting the Agent Identity Document. The AgentDocument status enum is updated to the four-state lowercase vocabulary the Lifecycle methods produce: active, suspended, retired, deprecated; the earlier capitalized Pending state and the inconsistent Revoked/ retired vocabulary are retired. The AGTP-LOG seriesinfo is updated to v02’s revision which finalizes the lifecycle event triggering methods to the eighteen- method floor.

11. *INSPECT read-access modes and lifecycle-method authorization specified normatively.* The INSPECT method definition (Section 7.2.4) now defines three normative read-access modes: public (default, unrestricted reads), agent_only (a caller authenticated as Agent-ID X may read only records whose subject is X), and operator_only (only operator-allowlisted client certificates may read). Access-control failures return 401 Unauthorized for unauthenticated callers and 403 Forbidden for authenticated callers whose identity is not permitted. A new Lifecycle Method Authorization subsection (Section 7.2.19) defines two authorization modes for ACTIVATE, DEACTIVATE, REINSTATE, REVOKE, and DEPRECATE: open (any caller; default for development only) and genesis_issuer (only the registrar whose key issued the agent's Agent Genesis, identified by mTLS certificate public-key fingerprint matching sha256(raw_ed25519_public_key_bytes(Agent_Genesis .issuer_public_key))). Lifecycle authorization failures return 401 / 403 by the same rules as INSPECT. Authorization for reads and writes is independent; operators *MAY* layer them (e.g., public reads with issuer-only writes). The Agent Genesis schema gains a new MUST-required field issuer_public_key carrying the Ed25519 public key of the signing issuer (32 raw bytes, base64url-encoded); verifiers re-verify signature against this key, and the SHA-256 fingerprint of the raw bytes is the Genesis-issuer fingerprint used for lifecycle-method authorization. The Method Summary table is updated to include 401 and 403 in the primary error codes for INSPECT and all five Lifecycle methods.

12. *Inline manifest signature on Agent Identity Document.* Three new RECOMMENDED fields are added to the Agent Identity Document: manifest_issuer (registrar identifier), manifest_issuer_public_key (Ed25519 public key, base64url), and manifest_signature (detached Ed25519 signature over the canonical document with manifest_signature excluded). Together they form a signed manifest. Conforming AGTP servers *MUST* verify the signature before serving the document. Verification confirms byte integrity; trust in the issuer key is a deployment policy (trusted-registrars list, CA trust anchors). The earlier stale forward reference to a signature envelope in [AGTP-CERT] is withdrawn — [AGTP-CERT] defines transport-layer cert binding, not a separate Identity Document envelope; inline signature fields fill the gap. The Identity Document Tamper-Proofing section (Section 6.3.5) is rewritten to describe the inline-signature model and to require registrars to strip operator-supplied signature fields before signing, preventing forged issuance. Unsigned manifests remain permitted for development and for deployments using transport-layer cert binding as the sole attestation path; production deployments *SHOULD* sign.

13. *DISCOVER built-in surface and lifecycle envelope format normatively settled.* Two changes that together finalize the eight-phase build’s read-side surface:

    (a) DISCOVER built-in endpoints are documented in [AGTP-API] as a path-keyed dispatch surface (DISCOVER /methods, /agents, /genesis, plus the new DISCOVER / directory and the optional /tools and /apis inventories). The reserved-paths rule forbids operator-authored endpoints from shadowing the built-in paths or registering under their first-segment prefixes. The legacy body-keyed DISCOVER target= form is now formally deprecated, with one-shot deprecation warning behavior and 400 Bad Request for path-versus-body conflicts. AGTP-API seriesinfo continues to be tracked from v08 normative refs.

    (b) INSPECT target=lifecycle response shape is tightened: each entry carries an explicit format field with value "jws" or "cose" reflecting the envelope used to sign the lifecycle event; servers *MAY* emit mixed-format streams; the configured envelope is selected by operator policy with jws as default and cose for SCITT-aligned deployments. The Lifecycle method specs are updated to specify that the signed envelope is either JWS Compact per [RFC7515] or COSE_Sign1 per [RFC9943], both normatively conformant. [AGTP-LOG] v02 gains a new Per-Agent Local Lifecycle Stream section specifying the envelope choice, the line-prefix disambiguation rule (jws: vs cose:), and the Audit-ID derivation for each envelope.

14. *Status codes 461 and 464 assigned.* The reserved codes 461 and 464 are assigned meanings to support the Runtime Contract Negotiation Substrate (RCNS) specified in [AGTP-API]: 461 RCNS Contract Available (confirm-first synthesis preview, response body carries a contract preview the caller may accept by re-issuing with the Contract-Synthesized header) and 464 RCNS No Contract (synthesis was attempted but no contract could be delivered, response body carries a structured reason from the vocabulary rcns-disabled, trust-tier-insufficient, composition-impossible, synthesis-error, contract-not-yours, contract-revoked). The reservations are made in this revision so that companion-draft RCNS work can target stable wire signatures; the full substrate specification (delivery modes, dispatcher gate, contract scoping, headers, Attribution-Record extensions) is in [AGTP-API]. Both the in-text status code table and the IANA Status Code Registry table are updated.

15. *RCNS dispatcher gate, observability, and lifecycle surfaces specified normatively.* Three new request headers are added to the Request Headers table: Allow-RCNS (caller opt-in to runtime negotiation), Contract-Synthesized (presentation of a previously- issued contract by synthesis_id), and Idempotency-Key (per-agent idempotency scope). Two new response headers are added: Contract-Synthesized (notification of optimistic-mode RCNS dispatch) and RCNS-Attempt-Id (handle for diagnostic INSPECT). The INSPECT method gains two new targets: target=contract (full record for a synthesized contract by synthesis_id) and target=rcns-attempt (diagnostic detail for failed synthesis attempts; the RCNS-Attempt-Id header on every 464 response names the attempt record). The REVOKE method gains a target=contract subsection (Section 7.2.16.1) for operator and originating-agent contract revocation with rcns_revoke lifecycle event emission. The SUSPEND method gains a synthesis_id parameter subsection (Section 7.2.12.1) for originating-agent self-release with rcns_release event emission. A new Contract-Related Lifecycle Events section (Section 6.5.7) reserves three RCNS event types on the per-agent lifecycle stream: rcns_propose_accepted, rcns_revoke, and rcns_release; these ride the same signed envelope stream as agent-state lifecycle events and are retrievable via INSPECT target=lifecycle. The complete RCNS specification — four-lock dispatcher gate, delivery modes, contract scoping, configuration knobs, observability surfaces — lives in [AGTP-API].

16. *Method aliases and HTTP gateway sidecar.* The Method Policy in [AGTP-API] gains an aliases field declaring a single-hop method-name translation map resolved ahead of catalog matching; the default seed maps the five legacy HTTP verbs to their canonical

AGTP counterparts (GET→FETCH, POST→CREATE, PUT→REPLACE, DELETE→REMOVE, PATCH→MODIFY). The original wire method is preserved on the Attribution-Record as requested_method per [AGTP-IDENTIFIERS]. A new HTTP Gateway Sidecar section (Section 8.5) specifies the translation contract for operators deploying a parallel HTTP listener alongside the AGTP daemon: HTTP method runs through the alias map, HTTP path is forwarded verbatim, the AGTP response is finalized through the standard path so attribution and audit chain hold, and Allow-RCNS is *MUST*-stripped at the gateway so HTTP callers cannot trigger runtime contract negotiation. The gateway is an adoption ramp, not part of the AGTP wire protocol; AGTP servers speaking to each other *MUST NOT* use the gateway as a transport hop.

17. *REVOKE target=stale-contracts subsection added; RCNS policy-change sweep wired through.* The REVOKE method gains a third parameter set (Section 7.2.16.2) for sweeping contracts whose captured recipe_version has drifted from the current loaded recipe version, pointing at the normative sweep semantics in [AGTP-API]. The Contract-Related Lifecycle Events table is updated: rcns_release is now triggered by either SUSPEND synthesis_id=... (agent self-release) or REVOKE target=stale-contracts (operator-fired eviction). When emitted by the sweep, the event payload *MUST* carry reason: policy-change-invalidation and the operator’s identifier so audit consumers can distinguish operator- fired invalidations from agent self-releases.

18. *Composition with External Identity Providers specified.* A new section (Section 8.6) documents three normative composition patterns for AGTP identity and external IdP credentials: Pattern 1 (AGTP identity only; closed-ecosystem default), Pattern 2 (AGTP identity plus an Authorization header carrying an OAuth bearer, OIDC id_token, or other IdP credential identifying the principal on whose behalf the agent acts), and Pattern 3 (OIDC-federated Genesis-issuer trust; specified in [AGTP-TRUST]). The Authorization request header is added to the request headers table with semantics referencing [RFC9110]. The 401 Unauthorized status code description is extended with a structured reason vocabulary that disambiguates external-credential failures (oauth-required, oauth-invalid) from AGTP-side authentication failures. The composition surface is opt-in via operator policy ([policies. oauth]); servers without OAuth configuration behave identically to pre-revision behavior. The raw Authorization header value and any token it carries *MUST NOT* appear on the Attribution-Record; only the validated, lifted claim (the acting_principal_id per [AGTP-IDENTIFIERS]) appears, preventing credential disclosure through audit-chain replay.

A.2. Wire Format Compatibility

The Agent-ID echo on responses was the deployed behavior in v07-conformant implementations; documenting it is editorial. The Attribution-Record change from "SHOULD when signing configured" to "MUST, with alg: none fallback" expands what v07-conformant servers without a signing key emit on the wire (they previously omitted the header; they now emit an alg: none JWS). v07 clients that ignored unsigned Attribution-Records continue to interoperate; clients that require cryptographic attestation *MUST* reject alg: none records per the updated security consideration.

Appendix B. Changes from v06

Version 07 confirms IANA registrations completed since v06, formalizes URI grammar and Identity Document terminology to match deployment, and makes architectural commitments that v06 prepared but did not lock.

B.1. Substantive Changes

The following substantive changes were made:

1. *IANA registrations confirmed.* The agtp:// URI scheme is registered under IANA per [RFC7595]; port 4480 is registered under unified service names agtp (TCP/TLS) and agtp-quic (QUIC)

per [RFC6335]. The IANA Considerations section (Section 11.1) and Stack Position section now state the registered values rather than "TBD." The pre-publication prohibition language has been removed.

2. *AGIS deprecated; AGTP-API introduced.* The Agentic Grammar and Interface Specification (AGIS) and the previously-proposed Agent Method Grammar (AMG) and AGTP-Methods drafts are deprecated and replaced by a single unified companion specification, AGTP-API [AGTP-API]. AGTP-API consolidates the method catalog, path grammar, endpoint primitive, semantic block, schema validation, server manifest format, per-server method policy (carried as the manifest's policies.methods sub-block), PROPOSE and synthesis semantics, and structural rejection codes (404, 405, 459, 460) into a single document because they describe a single concept: what makes a valid agent-server contract. The Method-Grammar header is removed; servers validate inbound method names against the AGTP-API approved verb list directly. The 454 Grammar Violation status code from earlier draft language is removed; method violations now return 459 (verb not in approved list) or 460 (path violates path grammar). The grammar-validation pathway is replaced by the AGTP-API contract validation pathway.

3. *Status code renumbering and new contract-level codes.* AGTP-specific status codes have been moved out of HTTP-assigned numeric ranges to avoid semantic collision with HTTP status codes that may appear in carried payloads. The mapping from v06: 451 → 455 (Scope Violation), 452 → 456 (Budget Exceeded), 453 → 457 (Zone Violation), 455 → 458 (Counterparty Unverified). New codes added per the AGTP-API contract model: 261 (Negotiation In Progress), 262 (Authorization Required; consolidates PROPOSE-time authorization, missing scope at endpoint dispatch, wildcards refused, and anonymous-discovery blocked), 263 (Proposal Approved), 405 (Method Not Allowed; method+path policy reject), 459 (Method Violation; method not in AGTP-API catalog), 460 (Endpoint Violation; path violates AGTP-API path grammar), 463 (Proposal Rejected). 408 (Timeout) and 410 (Gone) retain their HTTP code numbers and are registered with AGTP-specific semantics. The 461, 462, 464, 465, and 552-555 ranges are reserved for future AGTP expansion.

4. *Form 1a URI grammar.* A new canonical URI form agtp://{agent-id}@{host}[:{port}] (Form 1a) is introduced for direct addressing where the resolver does not yet have a path from canonical Agent-ID to endpoint. The Agent-ID identifies; the hostname is a resolution hint. Form 3 (path-style domain anchored) is retained as a compatibility form but deprioritized for new deployments.

5. *Port portion of URIs is OPTIONAL.* When the port is omitted from any AGTP URI, clients *MUST* use the IANA-assigned default port 4480.

6. *Wire-format framing.* A new normative subsection (Section 5.4.1) requires explicit Content-Length framing on every AGTP message and prohibits TLS socket-level half-close (shutdown(SHUT_WR)). This addresses a deployment-level gap surfaced by early implementations.

7. *Agent Manifest Document renamed to Agent Identity Document.* The artifact now matches the IANA-registered media type application/vnd.agtp.identity+json. The schema is enumerated in § 5.5 with field-by-field semantics: 16 REQUIRED fields, 14 RECOMMENDED fields, and 2 CONDITIONAL fields. The previous v06 schema is preserved in substance; the rename and the explicit field-level enumeration are the changes. The signature field is relocated to an envelope specified in [AGTP-CERT].

8. *Twelve-method protocol floor.* The core method set has been redesigned as a twelve-method floor organized into six cognitive verbs (QUERY, DISCOVER, DESCRIBE, SUMMARIZE, PLAN, PROPOSE) and six mechanics verbs (EXECUTE, DELEGATE, ESCALATE, CONFIRM, SUSPEND, NOTIFY). DISCOVER, PLAN, and EXECUTE are new core methods in v07. The v06 core methods BOOK, SCHEDULE, LEARN, and COLLABORATE are demoted to Tier 2 standard extended methods and are now catalogued in the AGTP-API method catalog ([AGTP-API]) rather than this document. EXECUTE absorbs the role earlier discussed for an INVOKE method: it is the generic carrier for application-layer payloads when higher-level frameworks are composed over AGTP.

9. *Composition with higher-level frameworks promoted to body text.* A new normative section (Section 8) establishes AGTP as a substrate for MCP, A2A, ACP, and ANP carried as content types inside AGTP method invocations. The substrate model, EXECUTE-as-carrier rule, precedence rule between AGTP headers and framework payloads, and canonical mapping table are normative in the body. The pre-existing AGMP composition appendix supplements the body section with worked wire examples.

10. *Trust score scoping.* The trust_score field (renamed from behavioral_trust_score) remains in the Identity Document REQUIRED schema. Computation methodology, freshness requirements, and signature binding are forward-referenced to a new companion specification, AGTP-TRUST [AGTP-TRUST], which will be developed separately.

11. *.well-known/agtp bootstrap convention.* A new subsection under § 5.1 specifies that organizations operating an AGTP namespace under a DNS domain SHOULD publish a bootstrap document at https://{domain}/.well-known/agtp per [RFC8615], declaring the organization's AGTP presence, endpoint, and verification anchors.

12. *Media type alignment.* The wire content type has been updated from application/agtp+json to application/vnd.agtp+json (vendor-tree pending standards-tree promotion at RFC publication). The Agent Identity Document uses application/vnd.agtp.identity+json. YAML variants are registered for both.

13. *Agent Genesis taxonomy clarification.* The permanent signed governance-layer origin document that establishes an agent's identity is named "Agent Genesis" throughout the specification. The taxonomy is: Agent Genesis (permanent signed origin document) → canonical Agent-ID (256-bit hash, used in all protocol operations) → Agent Certificate (optional X.509 v3 credential for TLS mutual authentication; see [AGTP-CERT]). The cross-layer relationship between Agent Genesis, canonical Agent-ID, and Agent Certificate has been clarified relative to earlier drafts where the origin document carried a different name.

14. *Runtime Contract Negotiation Substrate (RCNS) framing.* A new section (Section 5.3) makes explicit a property that earlier versions implied but did not name: AGTP fixes the protocol surface at twelve methods and negotiates any additional endpoint surface at runtime via PROPOSE governed by AGTP-API, completing in a single round-trip. The RCNS framing is added to the abstract and to Design Principles. The mechanism (PROPOSE method, AGTP-API contract validation, dynamic endpoint negotiation) was present in v06; v07 names the property and exposes it as a defining characteristic of the protocol.

B.2. Wire Format Compatibility

The status code renumbering and the rename from "Agent Manifest Document" to "Agent Identity Document" are wire-format-visible changes from v06. Implementations of v06 will require updates to emit and consume v07 codes and document type identifiers. Implementations following v06 may continue to interoperate with v07 servers that operate in a backward-compatibility mode that recognizes both old and new code numbers, but conformant v07 implementations MUST emit only the v07 codes.

B.3. Rationale

The v06 specification was prepared concurrently with IANA filings. v07 closes the loop on those filings (URI scheme, port, media types in evaluation) and propagates the architectural decisions that the filings made permanent. The status code renumbering is conservative: keeping 451 collided with RFC 7725 (Unavailable for Legal Reasons), which is an HTTP code that may legitimately appear in payloads carried over AGTP. The collision was harmless in v06 because no production implementations existed; it would not remain harmless as deployment grows.

The twelve-method floor reflects roughly two years of accumulated experience with the v00 through v06 method sets. The cognitive / mechanics organization is informational rather than normative, but provides a stable mental model that constrains future additions to the floor and guides extended-method design. EXECUTE absorbing INVOKE removes a naming ambiguity: EXECUTE is the clearer verb for the operation, and avoiding two methods in the carrier role simplifies composition with higher-level frameworks.

Appendix C. Authority-Scope Format

Authority-Scope values are expressed as a comma-separated list of scope tokens, each following the pattern domain:action or domain:* for full-domain access. Multi-segment scope tokens of the form domain:subdomain:action are admissible for namespaced operations (e.g., mcp:tools:execute). The encoding follows the HTTP-standard list-valued header convention [RFC9110]: tokens are separated by a comma, with optional surrounding whitespace permitted for readability. Tokens *MUST* be lowercase ASCII segments separated by colons, with no internal whitespace.

The wire-level grammar in ABNF:

```
Authority-Scope = scope-token *( OWS "," OWS scope-token )
scope-token     = scope-segment 1*( ":" scope-segment )
scope-segment   = 1*( ALPHA / DIGIT / "-" / "_" ) / "*"
OWS             = *( SP / HTAB )   ; optional whitespace, per RFC 9110
```

A scope token *MUST* contain at least two segments separated by a colon. The final segment is the action; preceding segments form the namespace path. The two-segment form domain:action is the canonical case; the three-or-more-segment form is reserved for deployments that need namespaced authority hierarchies (e.g., external-protocol bridging, hierarchical resource grants).

Parsers *MUST* accept any amount of optional whitespace adjacent to the comma separator (including none) and *MUST* treat "calendar:book,calendar:query" and "calendar:book, calendar:query" as equivalent. Implementations *SHOULD* emit a single space after the comma for human readability.

Examples:

```
Authority-Scope: calendar:book, calendar:query
Authority-Scope: documents:summarize, documents:query, knowledge:learn
Authority-Scope: *:query
Authority-Scope: booking:*, payments:confirm
Authority-Scope: mcp:tools:execute, knowledge:query
```

In the Agent Identity Document, Authority-Scope is represented as a JSON array of scope-token strings (one token per array element), not as a single comma-separated string. The wire-header encoding and the JSON-document encoding carry the same logical set of tokens.

Reserved domains (initial set):

Domain	Description
calendar	Scheduling and time-based resource management
documents	Document access, summarization, and annotation
knowledge	Agent context and memory operations
booking	Reservation and resource allocation
payments	Financial transactions and confirmations
agents	Delegation and collaboration with other agents
escalation	Escalation routing and handler management
activation	Governed agent package activation (ACTIVATE method extension)
discovery	Agent discovery and capability query operations (DISCOVER, DESCRIBE)
budget	Resource budget declaration and QUOTE pre-flight operations
telemetry	Telemetry export and observability operations
zone	Network zone boundary declaration and enforcement
suspend	Session suspension and resumption operations
merchant	Merchant identity resolution and counterparty verification (see [AGTP-MERCHANT])
intent	Intent Assertion issuance and validation (see [AGTP-MERCHANT])
*	All domains require explicit grant; use with caution

Table 46: Reserved Authority-Scope Domains

Appendix D. Example AGTP Wire Formats

The following examples use a human-readable pseudo-wire format with HTTP-style headers followed by a JSON body. The Content-Type for all AGTP message bodies is application/vnd.agtp+json.

D.1. QUERY Request and Response

```
AGTP/1.0 QUERY
Agent-ID: agt-7f3a9c2d
Authority-Scope: documents:query, knowledge:query
Session-ID: sess-a1b2c3d4
Task-ID: task-0042
Content-Type: application/vnd.agtp+json

{
  "task_id": "task-0042",
  "parameters": {
    "intent": "Key arguments against MCP re: HTTP overhead",
    "scope": ["documents:research", "knowledge:session"],
    "format": "structured",
    "confidence_threshold": 0.75
  }
}

AGTP/1.0 200 OK
Task-ID: task-0042
Server-ID: srv-knowledge-01
Attribution-Record: [signed attribution token]
Content-Type: application/vnd.agtp+json

{
  "status": 200,
  "task_id": "task-0042",
  "result": {
    "results": [{"content": "...", "source": "doc-agtp-research",
                 "confidence": 0.91}],
    "result_count": 1
  }
}
```

D.2. EXECUTE Request and Response (Carried Application Payload)

The following example shows EXECUTE carrying an application-layer booking action. The Authority-Scope on the request governs the action; a scope mismatch would return 455.

```
AGTP/1.0 EXECUTE
Agent-ID: agt-travel-planner
Authority-Scope: booking:*, calendar:book
Session-ID: sess-trip-2026-04
Task-ID: task-0107
Content-Type: application/vnd.agtp+json

{
  "method": "EXECUTE",
  "task_id": "task-0107",
  "parameters": {
    "action": "reserve-flight",
    "parameters": {
      "resource_id": "flight-AA2847",
      "principal_id": "usr-chris-hood",
      "time_slot": "2026-04-15T08:00:00Z",
      "options": {"seat_preference": "aisle", "class": "economy"}
    },
    "idempotency_key": "trip-2026-04-15-AA2847"
  }
}

AGTP/1.0 200 OK
Task-ID: task-0107
Attribution-Record: [signed attribution token]
Content-Type: application/vnd.agtp+json

{
  "status": 200,
  "task_id": "task-0107",
  "result": {
    "booking_id": "BK-2026-0107",
    "status": "confirmed",
    "resource_id": "flight-AA2847",
    "confirmation_code": "XQRT7Y"
  }
}
```

D.3. EXECUTE Carrying an MCP Tool Invocation

The following example shows EXECUTE carrying an MCP tool invocation. The carried payload's Content-Type identifies it as MCP; the AGTP server dispatches to its MCP handler. AGTP supplies identity, authority, and attribution at the wire level; MCP supplies tool semantics in the payload.

```
AGTP/1.0 EXECUTE
Agent-ID: agt-7f3a9c2d
Authority-Scope: mcp:tools:execute, knowledge:query
Session-ID: sess-mcp-bridge-01
Task-ID: task-0210
Content-Type: application/vnd.mcp.tools+json

{
  "method": "EXECUTE",
  "task_id": "task-0210",
  "parameters": {
    "action": "mcp.tools.call",
    "payload_type": "application/vnd.mcp.tools+json",
    "payload": {
      "tool": "search_documents",
      "arguments": {
        "query": "AGTP composition with higher-level frameworks",
        "limit": 5
      }
    }
  }
}

AGTP/1.0 200 OK
Task-ID: task-0210
Server-ID: srv-mcp-bridge
Attribution-Record: [signed attribution token]
Content-Type: application/vnd.mcp.tools+json

{
  "status": 200,
  "task_id": "task-0210",
  "result": {
    "tool_response": {
      "matches": [
        {"doc_id": "doc-0042", "score": 0.91, "excerpt": "..."}
      ]
    }
  }
}
```

D.4. ESCALATE Request and Response

```
AGTP/1.0 ESCALATE
Agent-ID: agt-procurement-03
Authority-Scope: booking:*, payments:confirm
Session-ID: sess-procurement-q2
Task-ID: task-0881
Content-Type: application/vnd.agtp+json

{
  "method": "ESCALATE",
  "task_id": "task-0881",
  "parameters": {
    "task_id": "task-0880",
    "reason": "scope_limit",
    "context": {
      "attempted_action": "EXECUTE",
      "resource": "vendor-contract-750k",
      "block_reason": "Exceeds agent authorization threshold"
    },
    "recipient": "usr-cfo",
    "deadline": "2026-03-19T09:00:00Z"
  }
}

AGTP/1.0 202 Accepted
Task-ID: task-0881
Server-ID: srv-escalation-handler
Content-Type: application/vnd.agtp+json

{
  "status": 202,
  "task_id": "task-0881",
  "result": {
    "escalation_id": "ESC-0881",
    "routed_to": "usr-cfo",
    "status": "pending_review",
    "task_paused": true,
    "estimated_review_by": "2026-03-19T09:00:00Z"
  }
}
```

Appendix E. Comparison Table

Criterion	AGTP	HTTP/ REST	gRPC	AGMP (MCP, A2A, ...)
Intent-native methods	Yes (12 Tier 1)	No	No	Partial

Intent semantics at protocol level	Native	None	None	Messaging layer only
Built-in agent identity	Yes	No	No	No
Authority scope enforcement	Protocol-level	None	None	Application-layer
Built-in attribution/ audit	Yes	No	No	Varies by impl.
Transport flexibility	TCP/UDP/ QUIC	TCP/TLS	HTTP/2	HTTP
Escalation as first-class primitive	Yes	No	No	No
Ecosystem maturity	Proposed	Mature	Mature	Emerging
Governance/ observability	Native	Manual/ bolt-on	Manual	Limited
Method registry extensibility	Yes (Expert Review)	Frozen (IETF Review)	N/A	N/A
Open core / royalty-free	Yes	Yes	Yes	Yes
Agent Identity Document	Native (.agtp format)	None	None	None
Tamper-proof identity surface	Yes (hash + signature)	No	No	No
Browser-accessible agent identity	Yes (read-only)	No	No	No

URI collision prevention	Domain-anchored	N/A	N/A	N/A
Agent Genesis	Yes (genesis record)	No	No	No
Domain-expiry lifecycle handling	Specified	N/A	N/A	N/A
Capability discovery	Native (DESCRIBE)	None	Reflection (partial)	None
Resource budget enforcement	Reserved for v01+ (Budget-Limit, 456)	None	None	None
Execution attestation (RATS)	Optional (RFC 9334)	None	None	None
Observability hooks	Reserved for v01+ (Telemetry-Export)	None	None	None
Network zone enforcement	Reserved for v01+ (AGTP-Zone-ID, 457)	None	None	None
Session suspension/ recovery	Native (SUSPEND method)	None	None	None
AGMP composition profiles	Normative body section	N/A	N/A	N/A

Table 47: AGTP Compared to Existing Approaches

HTTP’s method registry (registered with IETF Review per [RFC9110]) is effectively frozen for new semantic methods because any new HTTP method must be backward-compatible with existing HTTP infrastructure globally. AGTP’s Expert Review + published spec procedure enables

the protocol to evolve its method vocabulary as the agent ecosystem develops, without the backward-compatibility constraints of the HTTP method space.

Appendix F. Glossary

Agent: A software system that executes tasks, makes decisions, and takes actions without continuous human supervision per transaction.

AGMP (Agent Group Messaging Protocol): The collective term for higher-layer AI agent messaging standards that operate over AGTP as their transport substrate, including MCP, A2A, ACP, and ANP. AGMPs define what agents say. AGTP defines how those messages move. See Section 1.6.

Agent Genesis: The permanent, cryptographically signed origin document issued to an agent at registration time by a governance platform. The source from which the canonical Agent-ID is derived (256-bit hash). Issued once, permanently bound, never reissued. See Section 6.7.

Agent Transfer Protocol (AGTP): The application-layer protocol defined in this document, providing a dedicated transport environment for agent traffic.

Agent-ID: A unique identifier for a specific agent instance. Carried in the Agent-ID request header on non-anonymous AGTP requests, and in the agent_id field of the Agent Identity Document. In the base spec, derived from the Agent Genesis hash. With [AGTP-CERT], cryptographically bound to a verified identity.

Agent Identity Document: A signed application/vnd.agtp.identity+json document returned when an agtp:// URI is resolved. Derived from the agent's .agent or .nomo package. Contains identity, lifecycle state, trust tier, trust score, behavioral scope, methods, capabilities, and verification anchors. Never contains executable content. Schema enumerated in Section 5.5.

AGTP-Zone-ID: A network-zone boundary identifier declaring the scope within which a request must be processed. Reserved for v01+: not normatively specified as a request header in this revision. When the header is promoted to normative status, SEPs *MUST* enforce zone boundaries and return 457 Zone Violation if a DELEGATE request would route outside the declared zone.

Attribution Record: A signed, logged record of an agent action,

sufficient for audit and compliance purposes. *MAY* include RATS attestation evidence per [RFC9334] for hardware-rooted execution proof in high-stakes domains.

Authority-Scope: A declared set of permissions defining what actions an agent is authorized to take, expressed as comma-separated domain:action tokens per [RFC9110] list-valued header conventions. See Appendix C for the normative ABNF.

Budget-Limit: A request header declaring the maximum resource consumption the principal authorizes for a method invocation, expressed as comma-separated unit=value tokens from the IANA AGTP Budget Unit Registry. Exceeding the declared limit causes 456 Budget Exceeded. Reserved for v01+ per Section 5.5.

Delegation Chain: An ordered record of Agent-IDs representing the sequence of delegations that led to the current request.

DESCRIBE: An AGTP cognitive method returning the declared capabilities, supported modalities, method vocabulary, and versioned feature set of a specific agent endpoint. Used for pre-task negotiation.

DISCOVER: An AGTP cognitive method returning a list of candidate Agent-IDs matching specified criteria. Distinguished from QUERY: DISCOVER returns endpoints to talk to; QUERY returns data to consume.

EXECUTE: An AGTP mechanics method that invokes a specific action with parameters or carries an application-layer payload from a higher-level framework (MCP, A2A, ACP). The generic carrier method for composition.

PLAN: An AGTP cognitive method that produces an unexecuted sequence of actions toward a stated goal. The result is a plan the requesting agent or its principal reviews before committing to execution.

AGTP-API: The IETF companion specification [AGTP-API] that defines the contract layer for AGTP: the curated method catalog, path grammar, endpoint primitive, semantic block, schema validation, server manifest format, per-server method policy (the policies.methods sub-block of the manifest), and PROPOSE and synthesis semantics. AGTP-API is the authoritative source for what makes a valid agent-server contract over AGTP. AGTP-API supersedes the earlier AGIS draft and the proposed AGTP-Methods draft, consolidating their concerns into a single specification.

Contract: The protocol-level definition of what an AGTP interaction means: the verb (drawn from the AGTP-API method catalog), the path (conforming to AGTP-API path grammar), the semantic block, the input and output schemas, the authority requirements, and the composition rules that govern synthesis. Distinguished from middleware concerns ("plumbing"): rate limiting, observability, caching, custom authentication, and request transformation. The protocol governs contracts; middleware governs operations.

Identity-first architecture: The architectural commitment that the canonical Agent-ID (256-bit content-addressed identifier) is the authoritative identity primitive in AGTP. Hosting, DNS anchors, and other resolution paths are aliases.

ESCALATE: An AGTP method representing an agent’s intentional deferral of a decision or action to a human principal or higher-authority agent. A first-class method, not a failure code.

Governance Token: A signed, time-limited JWT artifact encoding a specific governance verdict for a specific action. The runtime companion to the Agent Genesis. Default TTL: 30 seconds. Must not be reused.

Intent Verb: An AGTP method name expressing the agent’s purpose, as distinguished from HTTP resource-operation verbs (GET, POST, PUT, DELETE).

Method Registry: The IANA-maintained registry of valid AGTP method names and their specifications. Registration requires Expert Review and a published specification.

Principal: The human, organization, or system that authorized an agent to act and is accountable for its actions.

Principal-ID: The identifier of the principal on whose behalf an agent operates. Carried in the agent identity document referenced by Agent-ID; not transmitted as a separate request header.

Scope-Enforcement Point (SEP): An AGTP-aware infrastructure component, load balancer, gateway, proxy, that enforces Authority-Scope and AGTP-Zone-ID compliance on AGTP requests without application-layer access. Requires [AGTP-CERT].

Scope Violation (455): An AGTP status code returned when an agent requests an action outside its declared Authority-Scope. A governance signal, not a protocol error. *MUST* be logged.

Session: An AGTP persistent connection context shared across

multiple method invocations within a single agent workflow.

SUSPEND (method): An AGTP Tier 1 core method that places a specific active session into a recoverable paused state, issuing a single-use base64url-encoded 128-bit resumption nonce. Session-scoped; does not affect registry lifecycle state. Category: ORCHESTRATE.

Trust Tier: A classification (1, 2, or 3) assigned to an agent at registration based on the strength of identity verification. Tier 1 requires one of three verification paths (DNS-anchored, log-anchored, or hybrid) and a .nomo governed package. Tier 2 is org-asserted without cryptographic verification. Tier 3 is experimental, not production-eligible.

551 Authority Chain Broken: An AGTP status code returned when one or more entries in the Delegation-Chain header cannot be verified as part of a valid and continuous delegation sequence. *MUST* be logged.

Appendix G. AGTP Composition with AGMPs

This appendix supplements Section 8 with worked wire examples for AGMP messages (MCP, A2A, ACP) carried over AGTP. The strategic positioning, substrate model, precedence rules, and canonical mapping table are normative in Section 8. Full composition specifications are provided in [AGTP-COMPOSITION].

G.1. Wire Example: A2A Task over AGTP

The following example shows an A2A task carried over AGTP DELEGATE. A2A task identity, message, and artifacts ride in the body; AGTP identity, authority, delegation chain, and attribution ride at the wire level.

```
AGTP/1.0 DELEGATE
Agent-ID: agtp://agtp.acme.tld/agents/orchestrator
Authority-Scope: agents:delegate, documents:query
Delegation-Chain: agtp://agtp.acme.tld/agents/orchestrator
Session-ID: sess-a1b2c3d4
Task-ID: task-0099
Content-Type: application/vnd.agtp+json

{
  "method": "DELEGATE",
  "task_id": "task-0099",
  "parameters": {
    "target_agent_id": "agtp://agtp.acme.tld/agents/analyst",
    "authority_scope": "documents:query",
    "delegation_token": "[signed token]",
    "task": {
      "a2a_task_id": "a2a-task-7f3a",
      "message": "Summarize Q1 financial reports",
      "artifacts": []
    }
  }
}
```

G.2. Wire Example: MCP Resource Fetch over AGTP

The following example shows an MCP resource fetch carried over AGTP QUERY. The fetch is read-only and naturally maps onto QUERY (rather than EXECUTE) because no application-side action is performed.

```
AGTP/1.0 QUERY
Agent-ID: agtp://agtp.acme.tld/agents/assistant
Authority-Scope: documents:query, knowledge:query
Session-ID: sess-mcp-b2c3d4
Task-ID: task-0100
Content-Type: application/vnd.agtp+json

{
  "method": "QUERY",
  "task_id": "task-0100",
  "parameters": {
    "intent": "fetch document corpus for Q1 financial analysis",
    "scope": ["documents:financial"],
    "modality": "mcp.resource",
    "mcp_resource_uri": "mcp://corpus/financial/q1-2026"
  }
}
```

For an MCP tool-call example using EXECUTE as the dispatch method, see the wire-format examples in Appendix D.

Author's Address

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