XEP-xxxx: XMPP Agent Gateway

Abstract

This specification defines an XMPP protocol extension for hosting, discovering, searching, and invoking addressable AI agents and tools based on and bridgeable to the Model Context Protocol Tool data model.

Author

Roman Shterenzon

Status

ProtoXEP

WARNING: This document has not yet been accepted for consideration or approved in any official manner by the XMPP Standards Foundation, and this document is not yet an XMPP Extension Protocol (XEP). If this document is accepted as a XEP by the XMPP Council, it will be published at <https://xmpp.org/extensions/> and announced on the <standards@xmpp.org> mailing list.

Type

Standards Track

Version

0.0.1 (2026-07-28)

Document Lifecycle

1. Experimental
2. Proposed
3. Stable
4. Final

Table of Contents

Appendices

1. Introduction

XMPP provides addressable identities, federation, routing, presence, and discovery that local processes, HTTP services, and MCP servers do not provide by themselves. This specification applies those properties to agent endpoints while denying federated visibility and invocation by default.

An endpoint has a bare JID, immutable versioned manifests, and tools based on and bridgeable to the MCP Tool data model. Clients use XMPP discovery, optional directory search, separately retrieved JSON Schemas, and a durable asynchronous task protocol.

The protocol is not an MCP JSON-RPC tunnel. Discovery, administration, durable tasks, MCP bridging, and human messaging are separate conformance profiles within this one document. A bridge preserves MCP objects where lossless and reports semantic loss otherwise.

This work grew out of real-world multi-agent orchestration. Openfire was used during development and integration testing.

2. Requirements

The protocol defined herein is designed to:

- give every hosted agent a stable, routable bare JID;
- allow a domain to host many virtual agents behind one internal or external component;
- enable standards-based discovery without requiring a proprietary registry;
- support both browse-oriented directory listing and user-oriented search;
- publish tool metadata based on and bridgeable to the MCP Tool data model without placing large JSON Schemas in every discovery response;
- support asynchronous, long-running, cancellable, and interactive tool executions;

- bind registration, discovery, and invocation to authenticated XMPP identities and deployment authorization policy;
- keep XMPP addressing, routing, tenant identity, and task correlation authoritative at the gateway boundary rather than delegating them to an LLM or tool runtime;
- provide deterministic correlation, version selection, schema validation, duplicate suppression, and terminal task semantics;
- keep optional human messaging separate from task execution and delivery state; and
- allow future extension to additional MCP capabilities without redefining the base protocol.

3. Use Cases

3.1 Discovering and Inspecting an Endpoint

Actors and success scenario: A user operates an XMPP client, the user's server advertises an agent gateway, and the gateway exposes a visibility-scoped directory. The client verifies the gateway feature, pages through visible endpoint items, and queries the selected endpoint's information as defined in Finding a Gateway, Listing Agent Endpoints, and Endpoint Information.

Alternate flows and errors: A candidate without the required feature is not a gateway. An empty page can mean that no endpoints are visible. A hidden or nonexistent endpoint produces the same safe error behavior defined in Error Handling; the client MUST NOT infer existence from a failed query.

3.2 Searching the Agent Directory

Actors and success scenario: A legacy-capable client and a gateway implementing the Jabber Search Compatibility profile exchange a search form and submission. The gateway applies the requester's visibility policy, orders matching endpoint JIDs deterministically, and returns an RSM page as defined in Agent Directory Search.

Alternate flows and errors: A gateway that advertises only Core Directory need not support search. An empty result can mean no visible match, and RSM counts include only visible endpoints. Malformed forms receive the applicable stanza error; hidden endpoints MUST NOT be distinguishable from absent results.

3.3 Discovering and Invoking a Tool

Actors and success scenario: A caller retrieves an endpoint's immutable manifest, records its exact version and hash, lists the tools pinned to that version, retrieves the selected tool's metadata and schemas, and sends an invocation to the target. The target validates and durably admits the task before

returning acceptance, following Manifest Retrieval, Listing Tools, Retrieving Tool Schemas, and Invocation.

Alternate flows and errors: A stale manifest hash produces a <conflict/> error, invalid arguments produce a <not-acceptable/> error, and an unknown or hidden tool or version produces an <item-not-found/> error. No task exists unless durable admission succeeds.

3.4 Recovering After Notification Loss

Actors and success scenario: A caller loses one or more lifecycle messages, reconnects on another authorized resource, and queries the target for complete task state. If the task is terminal and its result remains within the promised retention period, the caller retrieves the immutable terminal result as defined in Task Recovery.

Alternate flows and errors: A result query for a non-terminal task produces an <unexpected-request/> error. Nonexistent, expired, unauthorized, and cross-tenant task IDs all produce the same <item-not-found/> error and safe payload. Notification loss never establishes failure or permission to create a replacement task.

3.5 Supplying Interactive Input

Actors and success scenario: A target pauses a task in state `input_required`, records one pending structured request, and notifies the caller. The caller can recover that request from complete task state and submit one schema-valid answer from any authorized resource of its bare JID, following Interactive Input.

Alternate flows and errors: Invalid input produces a <not-acceptable/> error. Unknown requests produce an <item-not-found/> error, while stale, expired, already answered, or revision-conflicting input produces a <conflict/> error. An identical replay of an accepted answer is idempotent.

3.6 Cancelling a Running Task

Actors and success scenario: An authorized caller requests cancellation while the target owns a non-terminal task. The target atomically commits the request, moves the task to `cancelling`, and later settles it as `cancelled`, `completed`, or `failed`, as defined in Cancellation and Deadlines and Task States.

Alternate flows and errors: Cancellation is advisory, so successful work can still win the race. If terminal settlement commits first, the cancellation request produces a <conflict/> error. If cancellation commits first, later tool output cannot overwrite the terminal result.

3.7 Retrying Lost Admission

Actors and success scenario: A caller loses the admission IQ result and retries the same request ID with the same replay fingerprint during the advertised replay window. The target returns the original accepted task and does not execute the tool twice, following Invocation and Idempotency and Delegation.

Alternate flows and errors: Reusing the request ID with any fingerprint difference produces a <conflict/> error. Stream loss, timeout, or missing progress does not prove non-execution; the caller MUST retry identically or reconcile through Task Recovery rather than create a fresh side-effecting request.

3.8 Denying Federated Access

Actors and success scenario: A remote requester contacts a gateway or endpoint across an authenticated server-to-server route. Unless trusted local policy explicitly grants visibility or invocation, the gateway denies access without revealing whether the endpoint, tool, or task exists, as illustrated in Federated Invocation Example.

Alternate flows and errors: An operator can separately allow remote visibility and invocation for an identified domain or principal. Authentication alone grants neither. Denied, hidden, and nonexistent objects use equivalent safe errors, timing defenses, and rate limits according to Security Considerations.

3.9 Registering and Activating an Endpoint

Actors and success scenario: An authenticated C2S account self-registers an immutable manifest for its own bare JID, or a trusted administrator provisions a virtual endpoint. The gateway validates and stores the version, returns its canonical hash, and explicitly activates a selected version as defined in Registration Flow.

Alternate flows and errors: Self-registration for another JID is forbidden, and a component cannot claim that a virtual endpoint authenticated independently. Re-registering identical content is idempotent; different content under an existing version produces a <conflict/> error. Activation fails unless the exact registered version and hash exist and the sender is authorized.

3.10 Bridging an MCP Tool

Actors and success scenario: An MCP client communicates through a bridge that exposes an XMPP endpoint. The bridge preserves MCP Tool schemas, annotations, execution metadata, and complete CallToolResult objects where representable while mapping discovery, invocation, lifecycle, and recovery according to MCP Mapping.

Alternate flows and errors: When either protocol cannot represent a capability or state exactly, the bridge records and reports the semantic loss. It keeps XMPP RSM and MCP tool-list cursors separate and MUST NOT claim transparent MCP Tasks support when task identity, creation, polling, result retrieval, cancellation, or retention semantics are lost.

4. Glossary

Agent
An automated XMPP entity that can receive messages and expose one or more tools.

Agent Gateway
An XMPP service that hosts, routes for, registers, and advertises agent endpoints.

Agent Endpoint
The bare JID and versioned API manifest representing one hosted agent.

Manifest
A JSON document describing an agent, its capabilities, and its tools.

Tool
A callable capability compatible with the MCP Tool data model and addressable by name within an endpoint version.

Task
One durable execution of a tool, identified by a task ID and represented by a lifecycle state. A task is distinct from an ordinary XMPP conversation even when its result contains natural-language text.

Caller
The authenticated XMPP entity that starts a task.

Target
The agent endpoint responsible for executing a task.

Tenant
A deployment-defined authorization partition. Tenant identifiers are not globally meaningful and MUST NOT be trusted when supplied by an untrusted peer.

5. Design Considerations

Discovery uses Service Discovery (XEP-0030) [1], Result Set Management (XEP-0059) [2], and Service Discovery Extensions (XEP-0128) [3] so generic XMPP clients and caches can identify gateways, endpoints, and tools without a proprietary directory API. A compatibility profile additionally uses Jabber Search (XEP-0055) [4] because deployed clients already provide user-facing search forms for it.

Caller-to-target operations use IQ because each command requires a definite protocol response. The IQ result acknowledges durable admission only; execution remains asynchronous. Target-to-caller progress and state-change notifications use messages because accepted work can outlive an IQ exchange. Complete IQ state and result queries make notification loss recoverable. Message Delivery Receipts (XEP-0184) [5] remains unsuitable as an execution acknowledgement because it says nothing about semantic acceptance.

Endpoint identity is the XMPP bare JID itself. No additional URI scheme is defined because routing, comparison, discovery, and authorization already operate on that identifier.

JSON Containers (XEP-0335) [6] defines a generic JSON container but is Deferred and not recommended for production use. This specification therefore uses dedicated, self-describing JSON-bearing elements whose semantics and schemas are defined here.

6. Protocol Namespaces and Identifiers

Table 1: Namespaces defined by this specification

Purpose	Namespace or FORM_TYPE
Agent directory feature and node	`urn:xmpp:agent-directory:0`
Agent API XML	`urn:xmpp:agent-api:0`
Manifest retrieval feature	`urn:xmpp:agent-api:0#manifest`
Schema retrieval feature	`urn:xmpp:agent-api:0#schema`
Self-registration feature	`urn:xmpp:agent-api:0#self-register`
Administrative provisioning feature	`urn:xmpp:agent-api:0#admin`
Tool collection node	`urn:xmpp:agent-tools:0#VERSION`

Tool feature and node prefix	`urn:xmpp:agent-tool:0`
Agent endpoint feature	`urn:xmpp:agent-endpoint:0`
Endpoint information FORM_TYPE	`urn:xmpp:agent-endpoint-info:0`
Tool information FORM_TYPE	`urn:xmpp:agent-tool-info:0`
Task core	`urn:xmpp:agent-task:0`
Task progress	`urn:xmpp:agent-task:0#progress`
Task cancellation	`urn:xmpp:agent-task:0#cancel`
Task interactive input	`urn:xmpp:agent-task:0#input`

An endpoint identifier is its normalized bare JID. A resourcepart MUST NOT be present. Implementations MUST compare endpoint identifiers according to the XMPP address comparison rules and MUST NOT compare display strings or URI wrappers.

Endpoint API versions MUST match `[A-Za-z0-9._~-]{1,64}`, are case-sensitive opaque strings, and are compared exactly. Empty values and whitespace are forbidden. A tool name is the exact MCP name: it MUST be a nonempty JSON string, MUST contain only characters legal in XML 1.0 when carried on the wire, and is compared code point by code point without Unicode normalization. Implementations SHOULD follow MCP's recommended length and ASCII character set, but this protocol imposes no smaller character repertoire or maximum beyond the advertised manifest and stanza-size limits.

A tool collection node is formed by appending `#` and the exact API version to `urn:xmpp:agent-tools:0`. A tool node is formed by appending `#`, the exact API version, another `#`, and the unpadded base64url encoding from RFC 4648 Section 5 of the exact UTF-8 tool name to `urn:xmpp:agent-tool:0`. For example, `forecast.get` in version `1.4.0` has node `urn:xmpp:agent-tool:0#1.4.0#Zm9yZWNhc3QuZ2V0`. Decoders MUST reject padding, noncanonical encodings, invalid UTF-8, empty decoded names, and decoded names that violate the preceding limits.

7. Architecture and Hosting

7.1 Addressing Model

The gateway MUST have a domain JID, such as `agents.example`. Every hosted endpoint MUST have a bare JID, such as `weather@agents.example`. A gateway MAY host endpoints below its own component domain or another domain delegated to it by the XMPP server.

Agent endpoint JIDs are ordinary XMPP addresses from the perspective of remote entities. A server MUST route stanzas addressed to hosted endpoint JIDs to the responsible gateway. The endpoint need not correspond to a C2S account and need not maintain an independent XML stream.

7.2 Component Deployment

A gateway MAY be implemented as an internal server module, an Jabber Component Protocol (XEP-0114) [7] component, or another trusted service integration. The on-wire agent protocol is independent of that choice. An external component deployment MUST provision:

- the component domain or delegated agent domain;
- an authenticated component connection;
- routing for the virtual endpoint JIDs;
- an authorization policy for registration, directory visibility, and invocation;
- persistent manifest and task state if tasks must survive restart; and
- bounded reconnect, keepalive, IQ correlation, and task timeout behavior.

Remote component connections MUST use confidentiality and server authentication appropriate to the deployment. A plaintext component secret sent across an untrusted network is not acceptable. Operators SHOULD place the component on a local or mutually authenticated encrypted transport and SHOULD rotate component credentials.

The gateway SHOULD send XMPP Ping (XEP-0199) [8] pings after a period of stream inactivity and SHOULD reconnect with capped exponential backoff and jitter. Outstanding IQ requests MUST fail or time out when the component stream is lost; they MUST NOT remain pending indefinitely.

7.3 Conformance Profiles

This specification remains one document, but an implementation can claim one or more independently testable profiles:

Table 2: Conformance profiles

Profile	Required behavior
Core Directory	Gateway discovery, RSM-paged agent listing, and endpoint information.

Jabber Search Compatibility	Core Directory plus Jabber Search (XEP-0055) [4] and Result Set Management (XEP-0059) [2] search.
Tool Discovery	Core Directory plus version-pinned tool items, tool information, manifest retrieval, and schema retrieval.
Task Execution	Tool Discovery plus IQ invocation, durable acceptance, notifications, and complete state and result recovery.
Interactive Tasks	Task Execution plus the independently advertised progress, cancellation, and input features.
Self Registration	Authenticated C2S endpoint registration of immutable versions for the sender's own bare JID.
Administrative Provisioning	Trusted registration, activation, deactivation, inspection, and removal operations.
Human Messaging	The optional application profile in Human Messaging Profile; it is not required by any discovery or task profile.

A gateway MUST NOT advertise a feature it does not implement for the addressed entity. Service-discovery feature vars are authoritative. Endpoint-wide and tool-specific support are both required: effective support is true only when the endpoint advertises the feature and the selected tool declares support in its pinned manifest.

8. Service Discovery

8.1 Finding a Gateway

A client SHOULD query its server using Service Discovery (XEP-0030) [1] items. A server that exposes an agent gateway SHOULD include the gateway JID as an item. A client MUST then query the candidate JID with `disco#info` and MUST verify the `urn:xmpp:agent-directory:0` feature rather than inferring support from a subdomain name.

Example 1. Client discovers services on its server

```
<iq from='juliet@example.net/phone'
    id='services-1'
    to='example.net'
```

```
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>

<iq from='example.net'
    id='services-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='agents.example' name='Example Agent Gateway'/>
  </query>
</iq>
```

8.2 Gateway Information

The gateway MUST advertise an identity whose 'category' attribute is "automation" and whose 'type' attribute is "agent-gateway". It MUST advertise the protocol features it implements. This specification does not use `directory/user`, which denotes a directory of end users rather than agent endpoints.

Example 2. Gateway service discovery information

```
<iq from='juliet@example.net/phone'
    id='gateway-info-1'
    to='agents.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq from='agents.example'
    id='gateway-info-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='automation'
              name='Example Agent Gateway'
              type='agent-gateway'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='jabber:iq:search'/>
    <feature var='jabber:x:data'/>
    <feature var='urn:xmpp:agent-directory:0'/>
    <feature var='urn:xmpp:agent-api:0#admin'/>
```

```
    <feature var='urn:xmpp:ping'/>
  </query>
</iq>
```

8.3 Listing Agent Endpoints

A client lists addressable agents by sending a `disco#items` query to the gateway. The query MAY omit `node` or set it to `urn:xmpp:agent-directory:0`. Each returned item MUST contain the endpoint bare JID and SHOULD contain its human-readable name. Because each item is itself an addressable entity, the item MUST NOT set `node`.

The directory node MUST answer `disco#info` with an identity whose 'category' attribute is "automation" and whose 'type' attribute is "agent-directory", plus features `http://jabber.org/protocol/disco#info` and `http://jabber.org/protocol/disco#items`.

Directory listing MUST support Result Set Management (XEP-0059) [2]. Results MUST be ordered by normalized endpoint bare JID using octetwise ascending order, and that bare JID is the stable RSM item ID. The gateway MAY enforce a page maximum smaller than requested. RSM `count` MUST count only endpoints visible to the requester.

Example 3. Client lists agents

```
<iq from='juliet@example.net/phone'
    id='agents-1'
    to='agents.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='urn:xmpp:agent-directory:0'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>50</max>
    </set>
  </query>
</iq>

<iq from='agents.example'
    id='agents-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='urn:xmpp:agent-directory:0'>
```

```
    <item jid='weather@agents.example' name='Weather Agent'/>
    <item jid='calendar@agents.example' name='Calendar Agent'/>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first>calendar@agents.example</first>
      <last>weather@agents.example</last>
      <count>2</count>
    </set>
  </query>
</iq>
```

The gateway MUST apply its visibility policy before returning items. An empty result is not evidence that no agents exist; it can mean that no endpoints are visible to the requester.

8.4 Endpoint Information

A client obtains endpoint metadata by sending a node-less `disco#info` query to the endpoint bare JID. The endpoint MUST advertise an identity whose 'category' attribute is "automation" and whose 'type' attribute is "agent-endpoint", features `http://jabber.org/protocol/disco#info` and `http://jabber.org/protocol/disco#items`, the task and API features it implements, and an Service Discovery Extensions (XEP-0128) [3] result form with `FORM_TYPE` `urn:xmpp:agent-endpoint-info:0`. `urn:xmpp:agent-endpoint:0` is a service-discovery feature var only. It is not a disco node; an endpoint MUST return an <item-not-found/> error if it is used as one.

Table 3: Endpoint information fields

Field	Requirement	Description
`server_name`	REQUIRED	Stable implementation-oriented name.
`server_title`	REQUIRED	Human-readable display name.
`description`	OPTIONAL	Human-readable endpoint description.
`version`	REQUIRED	Selected manifest/API version.
`manifest_hash_algo`	REQUIRED	Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] algorithm token for the canonical manifest hash; this version requires `sha-256`.
`manifest_hash_value`	REQUIRED	Base64-encoded Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] hash value

		without an algorithm prefix.
`cold_start_supported`	REQUIRED	Data Forms (XEP-0004) [10] boolean indicating whether invocation can wake an endpoint whose runtime is not currently active. Responders MUST emit `1` or `0`.
`request_replay_seconds`	REQUIRED	Positive integer minimum admission replay window, measured from first receipt of a request ID.

Example 4. Endpoint service discovery information

```
<iq from='weather@agents.example'
    id='endpoint-info-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='automation'
              name='Weather Agent'
              type='agent-endpoint'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='urn:xmpp:agent-endpoint:0'/>
    <feature var='urn:xmpp:agent-api:0#manifest'/>
    <feature var='urn:xmpp:agent-api:0#schema'/>
    <feature var='urn:xmpp:agent-task:0'/>
    <feature var='urn:xmpp:agent-task:0#progress'/>
    <feature var='urn:xmpp:agent-task:0#cancel'/>
    <x xmlns='jabber:x:data' type='result'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:agent-endpoint-info:0</value>
      </field>
      <field var='server_name'><value>weather</value></field>
      <field var='server_title'><value>Weather Agent</value></field>
      <field var='description'><value>Provides weather forecasts.</value>
</field>
      <field var='version'><value>1.4.0</value></field>
      <field var='manifest_hash_algo'><value>sha-256</value></field>
      <field var='manifest_hash_value'>
<value>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</value></field>
      <field var='cold_start_supported'><value>1</value></field>
```

```
      <field var='request_replay_seconds'><value>86400</value></field>
    </x>
  </query>
</iq>
```

8.5 Listing Tools

A client lists an endpoint's tools by sending a `disco#items` query to the endpoint JID using the version-pinned collection node `urn:xmpp:agent-tools:0#VERSION`. Each item MUST use the endpoint JID, MUST identify the same version and tool with a tool node, and SHOULD include a human-readable name. Tool listing MUST support Result Set Management (XEP-0059) [2], ordered by the exact UTF-8 tool-name bytes in octetwise ascending order; the complete encoded tool node is the stable RSM item ID. Counts include only tools visible to the requester.

The collection node MUST answer `disco#info` with an identity whose 'category' attribute is "automation" and whose 'type' attribute is "agent-tool-collection", plus features `http://jabber.org/protocol/disco#info` and `http://jabber.org/protocol/disco#items`. Its version is fixed by the node identifier.

Example 5. Client lists endpoint tools

```
<iq from='juliet@example.net/phone'
    id='tools-1'
    to='weather@agents.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='urn:xmpp:agent-tools:0#1.4.0'>
    <set xmlns='http://jabber.org/protocol/rsm'><max>50</max></set>
  </query>
</iq>

<iq from='weather@agents.example'
    id='tools-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='urn:xmpp:agent-tools:0#1.4.0'>
    <item jid='weather@agents.example'
          name='Get Forecast'
          node='urn:xmpp:agent-tool:0#1.4.0#Zm9yZWNhc3QuZ2V0'/>
    <set xmlns='http://jabber.org/protocol/rsm'>
```

```
      <first>urn:xmpp:agent-tool:0#1.4.0#Zm9yZWNhc3QuZ2V0</first>
      <last>urn:xmpp:agent-tool:0#1.4.0#Zm9yZWNhc3QuZ2V0</last>
      <count>1</count>
    </set>
  </query>
</iq>
```

8.6 Tool Information

A client obtains tool metadata by querying its version-pinned tool node with `disco#info`. The response MUST advertise an identity whose 'category' attribute is "automation" and whose 'type' attribute is "agent-tool", features `http://jabber.org/protocol/disco#info` and `urn:xmpp:agent-tool:0`, the tool-specific task features it supports, and an Service Discovery Extensions (XEP-0128) [3] result form with `FORM_TYPE` `urn:xmpp:agent-tool-info:0`.

Table 4: Tool information fields

Field	Requirement	Description
`name`	REQUIRED	Stable tool name, unique within the endpoint version.
`title`	OPTIONAL	Human-readable title. A client displays `name` when this field is absent.
`description`	OPTIONAL	Human-readable tool behavior.
`api_version`	REQUIRED	Manifest/API version to which the tool belongs.
`input_schema_hash_algo`	REQUIRED	Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] algorithm token for the canonical input schema hash.
`input_schema_hash_value`	REQUIRED	Base64-encoded Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] input schema hash.
`output_schema_hash_algo`	OPTIONAL	Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] algorithm token for the canonical output schema hash.

`output_schema_hash_value`	OPTIONAL	Base64-encoded Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] output schema hash.
`read_only`	OPTIONAL	MCP `readOnlyHint`.
`destructive`	OPTIONAL	MCP `destructiveHint`.
`idempotent`	OPTIONAL	MCP `idempotentHint`.
`open_world`	OPTIONAL	MCP `openWorldHint`.

Each hash algorithm/value field pair MUST appear together. The value is the Base64 content that would appear in an Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] <hash/> element; it does not contain an algorithm prefix.

Annotations are optional untrusted hints and MUST NOT be used as authorization decisions. In particular, `read_only=true` does not prove that execution has no side effects. An absent field means unknown; a bridge MUST preserve absence and MUST NOT manufacture a value while claiming a lossless mapping.

Example 6. Client retrieves tool information

```
<iq from='juliet@example.net/phone'
    id='tool-info-1'
    to='weather@agents.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='urn:xmpp:agent-tool:0#1.4.0#Zm9yZWNhc3QuZ2V0'/>
</iq>

<iq from='weather@agents.example'
    id='tool-info-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='urn:xmpp:agent-tool:0#1.4.0#Zm9yZWNhc3QuZ2V0'>
    <identity category='automation' name='Get Forecast' type='agent-tool'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='urn:xmpp:agent-tool:0'/>
    <feature var='urn:xmpp:agent-task:0#progress'/>
```

```
    <feature var='urn:xmpp:agent-task:0#cancel'/>
    <x xmlns='jabber:x:data' type='result'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:agent-tool-info:0</value>
      </field>
      <field var='name'><value>forecast.get</value></field>
      <field var='description'><value>Gets a forecast for one city.</value>
</field>
      <field var='api_version'><value>1.4.0</value></field>
      <field var='input_schema_hash_algo'><value>sha-256</value></field>
      <field var='input_schema_hash_value'>
<value>XxGP62PwgOW9NLxx8ysunvuijMhSd0y4j1SSRTAbt7c=</value></field>
      <field var='read_only'><value>1</value></field>
      <field var='destructive'><value>0</value></field>
      <field var='idempotent'><value>1</value></field>
    </x>
  </query>
</iq>
```

8.7 Retrieving Tool Schemas

To avoid placing large JSON Schemas in discovery responses, a client retrieves each schema with an IQ of type "get" containing a <schema-request/> element. The 'tool', 'version', and 'direction' attributes are REQUIRED. The request MUST include the expected manifest hash. The response contains a <schema/> element with the exact version and media type plus Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] <hash/> child elements for the manifest and schema. After XML character extraction, the <json/> element's character data MUST be the exact RFC 8785 canonical UTF-8 bytes represented by the schema hash. A target MUST return a <conflict/> error when the supplied manifest hash does not select the immutable registered object.

Example 7. Client retrieves a tool input schema

```
<iq from='juliet@example.net/phone'
    id='schema-1'
    to='weather@agents.example'
    type='get'>
  <schema-request xmlns='urn:xmpp:agent-api:0'
                  direction='input'
                  tool='forecast.get'
                  version='1.4.0'>
    <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
```

```
  </schema-request>
</iq>

<iq from='weather@agents.example'
    id='schema-1'
    to='juliet@example.net/phone'
    type='result'>
  <schema xmlns='urn:xmpp:agent-api:0'
          direction='input'
          media-type='application/schema+json'
          tool='forecast.get'
          version='1.4.0'>
    <manifest-hash>
      <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
    </manifest-hash>
    <schema-hash>
      <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>XxGP62PwgOW9NLxx8ysunvuijMhSd0y4j1SSRTAbt7c=</hash>
    </schema-hash>
    <json>{"additionalProperties":false,"properties":{"city":
{"type":"string"}},"required":["city"],"type":"object"}</json>
  </schema>
</iq>
```

Tool schemas MUST be valid JSON Schema objects. Version 0 uses these rules:

- The recognized dialect is JSON Schema 2020-12; omission of `$schema` selects it.
- Core, Applicator, Validation, Meta-Data, Format-Annotation, Content, and Unevaluated vocabularies are required. `format` is annotation-only unless an extension explicitly requires supported assertion behavior.
- An input schema MUST have an object root.
- `$id` establishes the normal base URI, but automatic network retrieval is forbidden.
- Same-document `$ref`, `$anchor`, `$dynamicRef`, and `$dynamicAnchor` MUST be supported. External references require an explicit allowlist and resource budget.
- A vocabulary declared `true` in `$vocabulary` is required and MUST cause registration to fail when unsupported. An unsupported vocabulary declared `false` MAY be ignored only according to its optional semantics.
- Regular expressions use the ECMA-262-compatible semantics required by JSON Schema and MUST be resource bounded.

- Unknown annotation keywords MAY be retained and ignored; an unsupported assertion keyword MUST cause registration to fail.

The target's evaluation of the exact registered schema is authoritative.

8.8 Identity and Presence

An endpoint MAY publish a profile with vCard4 over XMPP (XEP-0292) [11]. For compatibility with deployed clients, it MAY also answer vcard-temp (XEP-0054) [12] requests. Profile information disclosed publicly MUST be limited to data intended for all requesters.

A hosted virtual endpoint SHOULD respond to presence subscription requests and probes as specified by XMPP IM. A gateway acting for a virtual endpoint MAY approve a subscription with `subscribed` and then send available presence. Presence reflects communications availability; detailed tool state remains in endpoint information.

Runtime availability is volatile and MUST be communicated with presence, not a cacheable service-discovery form. A cold-startable endpoint MAY be unavailable in XMPP presence while still advertising the static capability `cold_start_supported=1`.

9. Agent Directory Search

This section defines the optional Jabber Search Compatibility profile. A Core Directory implementation is not required to implement it. A conforming implementation of this profile MUST implement Jabber Search (XEP-0055) [4] and Result Set Management (XEP-0059) [2] at the gateway JID and advertise `jabber:iq:search`. It MUST support the legacy `nick` field and SHOULD support the Data Forms (XEP-0004) [10] extensibility profile.

Jabber Search (XEP-0055) [4] is a compatibility surface for existing XMPP directory-search user interfaces. It complements, rather than replaces, browse-oriented Service Discovery (XEP-0030) [1] listing.

The search form MUST contain a hidden `FORM_TYPE` field whose value is `jabber:iq:search`. The `nick` field matches a deployment-defined combination of agent localpart, stable name, and display title. Matching SHOULD be Unicode-aware and case-insensitive. An empty query MAY list all endpoints visible to the requester.

Example 8. Gateway returns an agent search form

```
<iq from='agents.example'
    id='search-form-1'
    to='juliet@example.net/phone'
    type='result'>
  <query xmlns='jabber:iq:search'>
    <instructions>Enter a nickname to search for matching agents.
</instructions>
    <nick/>
    <x xmlns='jabber:x:data' type='form'>
      <title>Agent Directory Search</title>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:search</value>
      </field>
      <field label='Nickname' type='text-single' var='nick'/>
    </x>
  </query>
</iq>
```

Example 9. Client submits an agent search

```
<iq from='juliet@example.net/phone'
    id='search-1'
    to='agents.example'
    type='set'>
  <query xmlns='jabber:iq:search'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>50</max>
    </set>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:search</value>
      </field>
      <field var='nick'><value>weather</value></field>
    </x>
  </query>
</iq>
```

Example 10. Gateway returns agent search results

```
<iq from='agents.example'
    id='search-1'
    to='juliet@example.net/phone'
    type='result'>
```

```
  <query xmlns='jabber:iq:search'>
    <x xmlns='jabber:x:data' type='result'>
      <reported>
        <field label='Jabber ID' type='jid-single' var='jid'/>
        <field label='Nickname' type='text-single' var='nick'/>
      </reported>
      <item>
        <field var='jid'><value>weather@agents.example</value></field>
        <field var='nick'><value>Weather Agent</value></field>
      </item>
    </x>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first>weather@agents.example</first>
      <last>weather@agents.example</last>
      <count>1</count>
    </set>
  </query>
</iq>
```

The gateway MUST apply the same visibility policy to search as to directory listing. It MUST return an empty result rather than reveal that hidden endpoints exist. Results MUST be ordered by normalized endpoint bare JID using octetwise ascending order; that bare JID is the stable RSM item ID. Counts include only visible matches. The RSM set is outside the Data Form, and the result form contains only <reported/> and <item/> child elements so that it conforms to the multi-item result rules in Data Forms (XEP-0004) [10].

10. Endpoint Registration

10.1 Manifest Model

An endpoint registers an immutable JSON manifest. The manifest is an independent XMPP data model based on the MCP Tool model and contains the following members:

Table 5: Agent manifest members

Member	Requirement	Description
`manifestSpecVersion`	REQUIRED	Manifest model version. This specification defines the JSON string `"0"`.

`agent.jid`	REQUIRED	Bare JID controlled by the registering principal.
`agent.name`	REQUIRED	Stable implementation name.
`agent.title`	OPTIONAL	Human-readable display name.
`agent.description`	OPTIONAL	Human-readable description.
`agent.version`	REQUIRED	Endpoint API version.
`implementation.name`	OPTIONAL	Implementation product name, distinct from endpoint identity.
`implementation.version`	OPTIONAL	Implementation release version, distinct from the endpoint API version.
`mcpProtocolVersion`	OPTIONAL	MCP protocol revision used by an MCP bridge.
`agent.vendor`	OPTIONAL	Provider or implementer name.
`agent.homepage`	OPTIONAL	Informational HTTPS URI. Gateways and clients MUST NOT dereference it as part of registration, discovery, or other automatic protocol processing.
`tools`	REQUIRED	Array of zero or more tool definitions.

Each tool MUST contain `name` and `inputSchema`. It MAY contain `title`, `description`, `outputSchema`, MCP `annotations`, MCP `execution`, MCP `_meta`, and an XMPP extension object named `urn:xmpp:agent-api:0`. The MCP members retain their MCP-defined semantics exactly. XMPP-specific progress, cancellation, timeout, authorization, and tag declarations belong only in the namespaced extension object. Tool names MUST satisfy the grammar in Protocol Namespaces and Identifiers and MUST be unique within one manifest.

The endpoint and tool service-discovery features are authoritative for XMPP capability negotiation. Tool-specific declarations apply only after that tool has been selected and MUST NOT override an endpoint capability advertised as unsupported.

Execution and authorization hints are declarations, not grants. A gateway MUST derive effective authorization from trusted policy and MUST fail closed when a declared permission or approval

requirement cannot be enforced.

This version uses all-or-nothing manifest visibility. A requester authorized to retrieve a manifest sees every tool in it. A deployment requiring tool-level visibility MUST publish separate endpoint versions or endpoint JIDs; it MUST NOT return a requester-filtered document under the registered manifest hash.

Example 11. Agent API manifest

```
{
  "manifestSpecVersion": "0",
  "agent": {
    "jid": "weather@agents.example",
    "name": "weather",
    "title": "Weather Agent",
    "description": "Provides weather forecasts.",
    "version": "1.4.0",
    "vendor": "Example"
  },
  "implementation": {"name":"example-weather","version":"3.2.1"},
  "mcpProtocolVersion": "2025-11-25",
  "tools": [{
    "name": "forecast.get",
    "title": "Get Forecast",
    "description": "Gets a forecast for one city.",
    "inputSchema": {
      "type": "object",
      "properties": { "city": { "type": "string" } },
      "required": ["city"],
      "additionalProperties": false
    },
    "outputSchema": {
      "type": "object",
      "properties": { "summary": { "type": "string" } },
      "required": ["summary"]
    },
    "annotations": {
      "readOnlyHint": true,
      "destructiveHint": false,
      "idempotentHint": true,
      "openWorldHint": true
    },
    "execution": {"taskSupport":"optional"},
```

```
    "urn:xmpp:agent-api:0": {
      "supportsProgress": true,
      "supportsCancellation": true,
      "defaultTimeoutSeconds": 30,
      "maximumTimeoutSeconds": 120,
      "requiredPermissions": ["weather.read"],
      "approvalRequired": false,
      "tags": ["weather", "forecast"]
    }
  }]
}
```

The following JSON Schema is normative for manifests. Limits shown here are protocol maxima; deployments MAY impose smaller documented limits. MCP-owned objects follow the MCP protocol revision declared by the manifest. Unknown top-level members are forbidden; extensions use URI-valued member names within a tool.

Normative agent manifest JSON Schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:xmpp:agent-api:0:manifest-json",
  "type": "object",
  "required": ["manifestSpecVersion", "agent", "tools"],
  "properties": {
    "manifestSpecVersion": {"const": "0"},
    "agent": {
      "type": "object",
      "required": ["jid", "name", "version"],
      "properties": {
        "jid": {"type":"string","minLength":3,"maxLength":3071},
        "name": {"type":"string","minLength":1,"maxLength":128},
        "title": {"type":"string","maxLength":256},
        "description": {"type":"string","maxLength":4096},
        "version": {
          "type":"string",
          "pattern":"^[A-Za-z0-9._~-]{1,64}$"
        },
        "vendor": {"type":"string","maxLength":256},
        "homepage": {"type":"string","format":"uri","maxLength":2048}
      },
      "additionalProperties": false
```

```
},
"implementation": {
  "type": "object",
  "required": ["name", "version"],
  "properties": {
    "name": {"type":"string","minLength":1,"maxLength":128},
    "version": {"type":"string","minLength":1,"maxLength":128}
  },
  "additionalProperties": false
},
"mcpProtocolVersion": {
  "type":"string",
  "pattern":"^[0-9]{4}-[0-9]{2}-[0-9]{2}$"
},
"tools": {
  "type": "array",
  "maxItems": 4096,
  "items": {
    "type": "object",
    "required": ["name", "inputSchema"],
    "properties": {
      "name": {
        "type":"string",
        "minLength":1
      },
      "title": {"type":"string","maxLength":256},
      "description": {"type":"string","maxLength":8192},
      "inputSchema": {"type":"object"},
      "outputSchema": {"type":"object"},
      "annotations": {"type":"object"},
      "execution": {"type":"object"},
      "_meta": {"type":"object"},
      "urn:xmpp:agent-api:0": {
        "type":"object",
        "properties":{
          "supportsProgress":{"type":"boolean"},
          "supportsCancellation":{"type":"boolean"},
          "supportsInput":{"type":"boolean"},
          "defaultTimeoutSeconds":{"type":"integer","minimum":1},
          "maximumTimeoutSeconds":{"type":"integer","minimum":1},
```

```
              "requiredPermissions":{
                "type":"array",
                "items":{"type":"string","minLength":1,"maxLength":256},
                "uniqueItems":true
              },
              "approvalRequired":{"type":"boolean"},
              "tags":{
                "type":"array",
                "items":{"type":"string","minLength":1,"maxLength":128},
                "uniqueItems":true
              }
            },
            "additionalProperties":false
          }
        },
        "patternProperties": {
          "^[a-z][a-z0-9+.-]*:": {}
        },
        "additionalProperties": false
      }
    }
  },
  "additionalProperties": false
}
```

10.2 Registration Flow

Self-registration and trusted administrative provisioning are different conformance profiles. Self-registration applies only to a genuine authenticated C2S endpoint whose authenticated bare JID equals `agent.jid`. A virtual endpoint behind an internal module or Jabber Component Protocol (XEP-0114) [7] component is provisioned by a trusted administrator through the Administrative Provisioning profile or a deployment-local API. Component authority MUST NOT be represented as if the virtual endpoint independently authenticated. Federation alone MUST NOT grant either authority.

Example 12. Authenticated account self-registers an immutable manifest

```
<iq from='weather@agents.example/runtime'
    id='register-1'
    to='agents.example'
    type='set'>
  <register xmlns='urn:xmpp:agent-api:0'>
    <manifest media-type='application/json'>
```

```
      {"manifestSpecVersion":"0",
       "agent":{"jid":"weather@agents.example",
                "name":"weather","version":"1.4.0"},
       "tools":[]}
    </manifest>
  </register>
</iq>

<iq from='agents.example'
    id='register-1'
    to='weather@agents.example/runtime'
    type='result'>
  <registered xmlns='urn:xmpp:agent-api:0'
              jid='weather@agents.example'
              version='1.4.0'>
    <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
  </registered>
</iq>
```

The gateway MUST validate the manifest before committing it. Before hashing, manifests and schemas MUST satisfy the I-JSON prerequisites of RFC 8785: no duplicate object names, valid Unicode with no lone surrogates, and numbers representable as IEEE-754 double precision. Strings are preserved without Unicode normalization. The gateway canonicalizes according to RFC 8785, hashes the canonical UTF-8 bytes with SHA-256, and represents the hash with an Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] <hash/> element using Base64 content. Invalid input is rejected before digest calculation. A hash is a cache validator and change detector, not proof of origin.

The tuple `(endpoint bare JID, API version)` is immutable. Re-registering the same version and hash is idempotent. Re-registering the same version with different canonical content MUST fail with a <conflict/> error. A new version creates a new immutable manifest. Semantic-version ordering MUST NOT be inferred.

The Administrative Provisioning profile MUST provide IQ operations to register an immutable version, activate an existing version, deactivate an endpoint, inspect registration status, remove an inactive version, and remove an endpoint. These operations use the same XML namespace, are addressed to the gateway, and require trusted administrative authorization. Activating a version changes only the mutable selected-version pointer; it does not alter a manifest. Removal MUST fail while a retained task or result depends on that version. A removed version leaves a permanent endpoint-scoped version-and-hash tombstone: identical content MAY be restored, but different content under that version MUST always fail

with a <conflict/> error. A deployment MAY expose equivalent local administration instead of advertising `urn:xmpp:agent-api:0#admin` federatively.

Table 6: Administrative provisioning operations

IQ set or get child	Successful IQ result child	Semantics
<register/>	<registered/>	Create an immutable version or idempotently confirm identical content.
<activate/> with 'jid' and 'version'	<active/>	Select an existing version after verifying its hash child.
<deactivate/> with 'jid'	<deactivated/>	Remove the active-version pointer without deleting manifests.
<registration-status/> with 'jid'	<registration/>	Return active version, registered versions and hashes, and whether each version is removable.
<remove-version/> with 'jid' and 'version'	<removed/>	Remove one inactive, unreferenced version.
<remove-endpoint/> with 'jid'	<removed/>	Deactivate and remove all versions only when none is retained by a task or result.

Example 13. Administrator activates a registered version

```
<iq from='admin@example.net/console'
    id='activate-1'
    to='agents.example'
    type='set'>
  <activate xmlns='urn:xmpp:agent-api:0'
            jid='weather@agents.example'
            version='1.4.0'>
    <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
  </activate>
</iq>

<iq from='agents.example'
    id='activate-1'
```

```
    to='admin@example.net/console'
    type='result'>
  <active xmlns='urn:xmpp:agent-api:0'
          jid='weather@agents.example'
          version='1.4.0'/>
</iq>
```

10.3 Manifest Retrieval

A client retrieves a manifest by sending an IQ of type "get" containing a <manifest-request/> element to the endpoint. Omitting the 'version' attribute requests the active version; a client MUST then pin the returned version and hash before listing tools, retrieving schemas, or invoking. Supplying the 'version' attribute requests that exact immutable version.

- The result MUST contain the exact version, a 'media-type' attribute whose value is "application/json", an Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] hash, and the canonical registered JSON document.
- After XML character extraction, the UTF-8 bytes in the <json/> element MUST be byte-for-byte identical to the RFC 8785 canonical bytes that were hashed and advertised.
- The stored source formatting is not protocol-visible and MUST NOT affect retrieval or hash validation.
- Access control MUST be identical to the corresponding tool and schema discovery policy.

A previously registered version remains addressable by exact value to a client that recorded it. Version enumeration is not defined. The target MUST retain the exact manifest and schemas selected at task admission through the task lifetime and promised result-retention period.

Example 14. Client retrieves the selected manifest

```
<iq from='juliet@example.net/phone'
    id='manifest-1'
    to='weather@agents.example'
    type='get'>
  <manifest-request xmlns='urn:xmpp:agent-api:0'/>
</iq>

<iq from='weather@agents.example'
    id='manifest-1'
    to='juliet@example.net/phone'
    type='result'>
  <manifest xmlns='urn:xmpp:agent-api:0'
            media-type='application/json'
            version='1.4.0'>
```

```
    <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
    <json>{"agent":
{"jid":"weather@agents.example","name":"weather","version":"1.4.0"},"manifest
SpecVersion":"0","tools":[]}</json>
  </manifest>
</iq>
```

11. JSON Payload Encoding

Every JSON-bearing element in this specification contains exactly one logical JSON text. Its XML character content, after normal XML entity decoding and concatenation of adjacent character-data nodes, MUST parse as one JSON value with no trailing non-whitespace data. Producers SHOULD emit one character-data node and MUST NOT depend on CDATA boundaries. Unless explicitly stated otherwise, the media type is `application/json`. Where the XML Schema requires a 'media-type' attribute, producers MUST include it even when `application/json` is the only permitted value.

JSON text MUST be encoded as UTF-8 and MUST also be legal XML character data. Producers MUST escape XML markup characters as XML requires, MUST encode JSON control characters with JSON escapes, and MUST reject lone Unicode surrogate code points and characters forbidden by the negotiated XML version. Consumers MUST preserve whitespace inside JSON strings but MAY ignore insignificant JSON whitespace outside strings.

Gateways MUST impose documented limits on JSON byte length, nesting depth, string length, array and object member count, and parsing time. External JSON Schema references MUST NOT be resolved unless an explicit allowlist and resource budget permit them.

12. Lifecycle JSON Schemas

The following JSON Schema is normative for task event payloads. It uses JSON Schema 2020-12 and selects a definition by the value of the 'type' attribute on the <event/> element. Unknown JSON members are forbidden except inside tool-result content, structured content, metadata, schemas, and deployment-qualified error details. Producers MUST use the exact camel-case member names shown here.

Normative task event payload JSON Schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:xmpp:agent-task:0:event-json",
```

```
"$defs": {
  "status": {
    "type": "object",
    "required": ["state", "updatedAt"],
    "properties": {
      "state": {"enum":["running","input_required","cancelling"]},
      "updatedAt": {"type":"string","format":"date-time"}
    },
    "additionalProperties": false
  },
  "progress": {
    "type": "object",
    "minProperties": 1,
    "properties": {
      "percent": {"type":"number","minimum":0,"maximum":100},
      "stage": {"type":"string","maxLength":256},
      "message": {"type":"string","maxLength":4096}
    },
    "additionalProperties": false
  },
  "input_required": {
    "type": "object",
    "required": ["requestId", "question", "inputSchema", "createdAt"],
    "properties": {
      "requestId": {
        "type":"string",
        "pattern":"^[A-Za-z0-9._~-]{22,128}$"
      },
      "question": {"type":"string","minLength":1,"maxLength":4096},
      "inputSchema": {"type":"object"},
      "createdAt": {"type":"string","format":"date-time"},
      "expiresAt": {"type":"string","format":"date-time"}
    },
    "additionalProperties": false
  },
  "toolResult": {
    "type": "object",
    "required": ["content"],
    "properties": {
      "content": {"type":"array","items":{"type":"object"}},
```

```
    "structuredContent": {},
    "isError": {"type":"boolean"},
    "_meta": {"type":"object"}
  },
  "additionalProperties": false
},
"completed": {
  "type": "object",
  "required": ["result"],
  "properties": {
    "result": {"$ref":"#/$defs/toolResult"},
    "summary": {"type":"string","maxLength":4096}
  },
  "additionalProperties": false
},
"failed": {
  "type": "object",
  "required": ["error"],
  "properties": {
    "error": {
      "type":"object",
      "required":["code","message","retryable"],
      "properties":{
        "code":{"type":"string","pattern":"^[A-Za-z0-9_.:-]{1,128}$"},
        "message":{"type":"string","minLength":1,"maxLength":4096},
        "retryable":{"type":"boolean"},
        "details":{"type":"object"}
      },
      "additionalProperties":false
    }
  },
  "additionalProperties": false
},
"cancelled": {
  "type": "object",
  "properties": {
    "reason": {"type":"string","maxLength":4096}
  },
  "additionalProperties": false
}
```

```
  }
}
```

A status event validates against `$defs.status`, and equivalently for `progress`, `input_required`, `completed`, `failed`, and `cancelled`. Every `date-time` string MUST also conform to XMPP Date and Time Profiles (XEP-0082) [13]. The value of the <input/> child element of <provide-input/> is validated against the exact pending `inputSchema`. A <task-result/> element contains the same object as the corresponding terminal event payload. Implementations MUST reject event payloads that do not validate.

13. Tool Execution

13.1 Invocation

Tool execution is asynchronous even when a local API offers a blocking convenience call. The caller creates a globally unpredictable request ID and sends an IQ of type "set" containing an <invoke/> element. The target creates the task ID only after successful validation and durable admission. Both identifiers MUST contain 22 to 128 ASCII characters from `[A-Za-z0-9._~-]` and MUST contain at least 128 bits of unpredictability generated by a cryptographically secure random number generator. A deployment-local correlation ID MAY associate this operation with application state, but it has no wire authority.

Example 15. Caller invokes a remote tool

```
<iq from='planner@agents.example/worker'
    id='invoke-1'
    to='weather@agents.example'
    type='set'>
  <invoke xmlns='urn:xmpp:agent-task:0'
          api-version='1.4.0'
          request-id='req-91a6d2f49c514f44a2d223f0f97bd9fe'
          tool='forecast.get'>
    <manifest-hash>
      <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
    </manifest-hash>
    <arguments media-type='application/json'>{"city":"Riga"}</arguments>
    <deadline>2026-07-27T19:15:00Z</deadline>
  </invoke>
</iq>
```

```
<iq from='weather@agents.example'
    id='invoke-1'
    to='planner@agents.example/worker'
    type='result'>
  <accepted xmlns='urn:xmpp:agent-task:0'
            created='2026-07-27T19:14:01Z'
            request-id='req-91a6d2f49c514f44a2d223f0f97bd9fe'
            retain-until='2026-07-28T19:14:01Z'
            revision='0'
            task-id='task-7f2a9c41d8804e96bb478af6453c0371'/>
</iq>
```

The 'request-id', 'tool', and 'api-version' attributes, manifest hash, and <arguments/> element are REQUIRED. The <arguments/> element MUST include a 'media-type' attribute whose value is "application/json". A target MUST pin the exact manifest and schemas before admission. The authoritative authorization principal is the immediate authenticated sender's bare JID. The full JID that issued the IQ is the default best-effort notification route; any authorized resource of the same bare JID can use recovery operations.

The target computes a replay fingerprint over the authenticated caller bare JID, target bare JID, protocol namespace, request ID, tool, API version, manifest hash, canonical arguments, deadline, and every security-relevant extension. During the advertised replay window, reuse with the same fingerprint MUST return the same accepted task and MUST NOT execute twice; reuse with a different fingerprint MUST fail with a <conflict/> error. A target MUST retain the replay record through the longer of that window, the task lifetime, and the promised result-retention period.

After the advertised replay window and all required task retention have expired, a target MAY forget the request ID and MAY admit a repeated request as new work. Therefore a caller MUST NOT retry after that window without tool-specific reconciliation. This protocol provides at-most-once task admission only inside the replay-retention window and does not claim exactly-once external side effects.

An IQ result containing an <accepted/> element represents durable task acceptance, not merely receipt of the request. The target MUST persist the task and replay record before sending that result.

- Malformed, unauthorized, conflicting, unavailable, expired, or schema-invalid requests fail before admission with an ordinary IQ error and create no task.
- Loss of the IQ result leaves execution uncertain; the caller MUST retry the identical request ID rather than create a new request.
- Message Delivery Receipts (XEP-0184) [5] does not apply to IQ admission.

13.2 Federated Invocation Example

This example shows a caller at `example.com` discovering and invoking an endpoint at `agents.example`. Normal XMPP server-to-server authentication establishes the immediate sender domain; the target's explicit federation policy still determines visibility and authorization.

Example 16. Federated discovery, durable acceptance, and completion

```
<iq from='alice@example.com/laptop'
    id='federated-disco-1'
    to='agents.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq from='agents.example'
    id='federated-disco-1'
    to='alice@example.com/laptop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='automation' type='agent-gateway'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='urn:xmpp:agent-directory:0'/>
  </query>
</iq>

<iq from='alice@example.com/laptop'
    id='federated-invoke-1'
    to='weather@agents.example'
    type='set'>
  <invoke xmlns='urn:xmpp:agent-task:0'
          api-version='1.4.0'
          request-id='req-f83bf930e7ae45e6bddca1afb566df08'
          tool='forecast.get'>
    <manifest-hash>
      <hash xmlns='urn:xmpp:hashes:2'
            algo='sha-256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=
</hash>
    </manifest-hash>
    <arguments media-type='application/json'>{"city":"Riga"}</arguments>
  </invoke>
```

```
</iq>

<iq from='weather@agents.example'
    id='federated-invoke-1'
    to='alice@example.com/laptop'
    type='result'>
  <accepted xmlns='urn:xmpp:agent-task:0'
            created='2026-07-28T10:00:00Z'
            request-id='req-f83bf930e7ae45e6bddca1afb566df08'
            retain-until='2026-07-29T10:00:00Z'
            revision='0'
            task-id='task-c27daf5e3a9b45e086a6616e28139dcb'/>
</iq>

<message from='weather@agents.example'
         id='federated-result-1'
         to='alice@example.com/laptop'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
         event-id='evt-f53573cb922d44be97a674971aa9da71'
         revision='1'
         task-id='task-c27daf5e3a9b45e086a6616e28139dcb'
         type='completed'>
    {"result":{"content":[{"type":"text","text":"Light rain, 18 C"}],
               "structuredContent":{"summary":"Light rain, 18 C"},
               "isError":false}}
  </event>
</message>
```

13.3 Lifecycle Events

The target sends best-effort lifecycle notifications as normal messages containing an <event/> element. The <event/> element's character data is a JSON object governed by Lifecycle JSON Schemas. Every event carries the receiver-generated task ID, a globally unique unpredictable event ID, and a global revision. Revision `0` is the accepted state returned by IQ. Each committed state transition MUST increase the revision by exactly one and generate exactly one event at that revision. Each committed progress update likewise increases it by exactly one. Delivery can lose an event, but the target does not create revision gaps through hidden transitions.

Table 7: Task lifecycle event types

Type	Sender	Required or common JSON members
`status`	Target	`state` and `updatedAt`.
`progress`	Target	`percent`, `stage`, and `message`, as applicable.
`input_required`	Target	`requestId`, `question`, `inputSchema`.
`completed`	Target	`result` containing the complete result envelope and optional `summary`.
`failed`	Target	`error` containing `code`, `message`, `retryable`, and optional qualified details.
`cancelled`	Target	Empty object or optional `reason`.

Example 17. Target notifies the caller that execution is running

```
<message from='weather@agents.example'
         id='status-task-7f2a9c41d8804e96bb478af6453c0371'
         to='planner@agents.example/worker'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
         event-id='evt-00d7b9a9b0864dcf9308d56e8f54b392'
         revision='1'
         task-id='task-7f2a9c41d8804e96bb478af6453c0371'
         type='status'>
    {"state":"running","updatedAt":"2026-07-27T19:14:02Z"}
  </event>
</message>
```

The value of the 'retain-until' attribute on the <accepted/> element is a unilateral promise by the target. Complete state and any terminal result MUST remain queryable until then, and state MUST remain queryable throughout a longer non-terminal lifetime. A caller requiring longer retention MUST preserve its own record.

When present, progress `percent` MUST be a finite number from 0 through 100 inclusive and SHOULD be nondecreasing. An identical replay MUST retain the same event ID, revision, type, and canonical payload. A caller MUST ignore such a replay. A repeated revision with a different event ID or payload is a

protocol error and MUST be ignored and logged. A revision gap does not imply failure; the caller SHOULD query task state, which returns the current revision and complete resumable state.

Example 18. Target reports progress

```
<message from='weather@agents.example'
         id='progress-task-7f2a9c41d8804e96bb478af6453c0371'
         to='planner@agents.example/worker'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
         event-id='evt-50a26b5692644a7880caa7ca56ca5b54'
         revision='2'
         task-id='task-7f2a9c41d8804e96bb478af6453c0371'
         type='progress'>
    {"percent":50,"stage":"fetch","message":"Forecast received"}
  </event>
</message>
```

Example 19. Target requests input and caller answers

```
<message from='weather@agents.example'
         id='input-request-task-7f2a9c41d8804e96bb478af6453c0371'
         to='planner@agents.example/worker'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
         event-id='evt-0b5851650e8a427f9b00499085d96f0f'
         revision='3'
         task-id='task-7f2a9c41d8804e96bb478af6453c0371'
         type='input_required'>
    {"requestId":"input-a5128a63d64146d5a6ffac65bb20c348",
     "question":"Which forecast unit should be used?",
     "inputSchema":{"type":"object","properties":{"unit":{"enum":["C","F"]}},
                    "required":["unit"]},
     "createdAt":"2026-07-27T19:14:04Z"}
  </event>
</message>

<iq from='planner@agents.example/worker'
    id='input-answer-1'
    to='weather@agents.example'
    type='set'>
  <provide-input xmlns='urn:xmpp:agent-task:0'
                 expected-revision='3'
```

```
                request-id='input-a5128a63d64146d5a6ffac65bb20c348'
                task-id='task-7f2a9c41d8804e96bb478af6453c0371'>
    <input media-type='application/json'>{"unit":"C"}</input>
  </provide-input>
</iq>

<iq from='weather@agents.example'
    id='input-answer-1'
    to='planner@agents.example/worker'
    type='result'>
  <input-accepted xmlns='urn:xmpp:agent-task:0'
                  revision='4'
                  task-id='task-7f2a9c41d8804e96bb478af6453c0371'/>
</iq>
```

Example 20. Caller requests cancellation and target confirms

```
<iq from='planner@agents.example/worker'
    id='cancel-request-1'
    to='weather@agents.example'
    type='set'>
  <cancel xmlns='urn:xmpp:agent-task:0'
          expected-revision='4'
          task-id='task-7f2a9c41d8804e96bb478af6453c0371'>
    <reason>Trip was cancelled</reason>
  </cancel>
</iq>

<iq from='weather@agents.example'
    id='cancel-request-1'
    to='planner@agents.example/worker'
    type='result'>
  <cancel-accepted xmlns='urn:xmpp:agent-task:0'
                   revision='5'
                   task-id='task-7f2a9c41d8804e96bb478af6453c0371'/>
</iq>

<message from='weather@agents.example'
         id='cancelled-task-7f2a9c41d8804e96bb478af6453c0371'
         to='planner@agents.example/worker'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
```

```
        event-id='evt-7413f189224b45bc8634f2eaec99961b'
        revision='6'
        task-id='task-7f2a9c41d8804e96bb478af6453c0371'
        type='cancelled'>{}</event>
</message>
```

Example 21. Target reports task failure

```
<message from='weather@agents.example'
         id='failed-task-7f2a9c41d8804e96bb478af6453c0371'
         to='planner@agents.example/worker'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
         event-id='evt-59dd231300a54913815812a0d6a422a0'
         revision='4'
         task-id='task-7f2a9c41d8804e96bb478af6453c0371'
         type='failed'>
    {"error":{"code":"unavailable",
              "message":"Forecast provider is temporarily unavailable.",
              "retryable":true}}
  </event>
</message>
```

Example 22. Target completes a task

```
<message from='weather@agents.example'
         id='completed-task-7f2a9c41d8804e96bb478af6453c0371'
         to='planner@agents.example/worker'
         type='normal'>
  <event xmlns='urn:xmpp:agent-task:0'
         event-id='evt-2c1187df307a45e6bdef7c3b34073c7c'
         revision='4'
         task-id='task-7f2a9c41d8804e96bb478af6453c0371'
         type='completed'>
    {"result":{"content":[{"type":"text","text":"Light rain, 18 C"}],
               "structuredContent":{"summary":"Light rain, 18 C"},
               "isError":false},
     "summary":"Forecast ready"}
  </event>
</message>
```

The `result` member is the complete tool result envelope. An MCP bridge preserves `CallToolResult.content`, `structuredContent`, `isError`, and `_meta` as a whole. An MCP result

with `isError=true` is still a successfully completed tool call and MUST NOT be converted into infrastructure state `failed`. When the selected tool declares an `outputSchema`, the target MUST validate `structuredContent` before completion, and the caller SHOULD validate it again.

13.4 Interactive Input

When a target cannot continue without caller input, it commits state `input_required` and sends a notification with a task-unique request ID, a human-readable question, a JSON Schema, creation time, and optional expiry time. At most one input request may be pending per task. Only an authorized resource of the authenticated caller bare JID may answer with an IQ of type "set" containing a <provide-input/> element. The target authoritatively validates the input and expected revision before returning an <input-accepted/> element. Unknown request IDs receive an <item-not-found/> error; schema-invalid input receives a <not-acceptable/> error; stale, expired, already answered, or conflicting input receives a <conflict/> error. Repeating an identical accepted IQ is idempotent.

Every state response while input is pending MUST include the complete pending request, including its expiry if any, so loss of the notification is recoverable. On input expiry the target MUST atomically issue a new input request or settle the task. Cancellation may transition pending input to `cancelling`. Terminal settlement wins any race with input, and input received after terminal settlement MUST fail with a <conflict/> error.

Interactive input is data entry, not a standardized authorization grant. Human approval in version 0 can be satisfied only through trusted local or out-of-band policy. A target that cannot verify a required approval MUST fail closed. Secrets SHOULD be provided through a credential mechanism rather than copied into task payloads.

13.5 Cancellation and Deadlines

Only an authorized resource of the authenticated caller bare JID may request cancellation using an IQ of type "set". A cancellation request is accepted exactly once, moves a non-terminal task to `cancelling`, and remains advisory until the target settles it as `cancelled`, `completed`, or `failed`. The target alone chooses and commits the terminal state. If settlement committed first, cancellation fails with a <conflict/> error; if cancellation committed first, later tool output cannot overwrite the terminal result.

Deadlines MUST conform to XMPP Date and Time Profiles (XEP-0082) [13]. If the target's accepted deadline expires while its task is non-terminal, the target MUST settle the task as `failed` with code `deadline-expired`. A caller-side wait timeout is only a local observation and MUST NOT create or assert a remote terminal state. Timeout does not prove that side effects did not occur; the caller MUST recover state or retry the identical request ID before considering a new side-effecting invocation.

13.6 Task States

Table 8: Task states

State	Meaning
`accepted`	Task was durably accepted.
`running`	Tool execution is in progress.
`input_required`	Target is waiting for caller input.
`cancelling`	Cancellation was requested but is not terminal.
`cancelled`	Target confirmed cancellation. Terminal.
`failed`	Execution failed. Terminal.
`completed`	Execution completed successfully. Terminal.

Table 9: Normative task transitions

Source	Destination	Authorizing action	Notification
`accepted`	`running`	Target starts execution.	`status`
`accepted` or `running`	`input_required`	Target commits one pending input request.	`input_required`
`input_required`	`running`	Target accepts valid caller input.	`status`
Any non-terminal state	`cancelling`	Target accepts an authorized cancellation IQ.	`status`
Any non-terminal state	`completed`	Target atomically commits a successful tool result.	`completed`
Any non-terminal state	`failed`	Target atomically commits an execution failure or deadline expiry.	`failed`

Any non-terminal state	`cancelled`	Target atomically confirms cancellation.	`cancelled`

Validation, rejection, queueing, startup, and caller-side timeouts are local conditions, not interoperable task states. A task exists only after durable acceptance. The permitted transitions are: `accepted` to `running`, `input_required`, `cancelling`, or any terminal state; `running` to `input_required`, `cancelling`, or any terminal state; `input_required` to `running`, `cancelling`, or any terminal state; and `cancelling` to any terminal state. Terminal states are immutable.

13.7 Task Recovery

A caller recovers state with an IQ of type "get" containing a <task-state-request/> element and retrieves a terminal result with a separate <task-result-request/> element.

- The target MUST authorize every operation against the caller bare JID.
- Syntactically invalid identifiers receive a <bad-request/> error.
- Nonexistent, expired, unauthorized, and cross-tenant task IDs MUST all receive the same <item-not-found/> error and safe payload.
- Those cases SHOULD use similar response timing and identical rate-limiting classes so they remain observationally equivalent.

The <task-state/> element MUST return the target endpoint bare JID, exact API version and manifest hash, state, current global revision, creation and update times, retention deadline, and deadline if any. While input is pending it MUST include a <pending-input/> element with a 'media-type' attribute whose value is "application/json" and character data containing the complete `input_required` object. A terminal state MUST identify whether a retained result is available. The <task-result/> element is valid only for a terminal task, returns the immutable terminal state and complete retained payload, and MUST fail with an <unexpected-request/> error while non-terminal. Notifications are advisory; these two IQ operations are the recovery source of truth.

Example 23. Caller recovers task state

```
<iq from='planner@agents.example/worker'
    id='task-state-1'
    to='weather@agents.example'
    type='get'>
  <task-state-request xmlns='urn:xmpp:agent-task:0'
                      task-id='task-7f2a9c41d8804e96bb478af6453c0371'/>
</iq>
```

```
<iq from='weather@agents.example'
    id='task-state-1'
    to='planner@agents.example/worker'
    type='result'>
  <task-state xmlns='urn:xmpp:agent-task:0'
              api-version='1.4.0'
              created='2026-07-27T19:14:01Z'
              endpoint='weather@agents.example'
              result-available='false'
              retain-until='2026-07-28T19:14:01Z'
              revision='2'
              state='running'
              task-id='task-7f2a9c41d8804e96bb478af6453c0371'
              updated='2026-07-27T19:14:03Z'>
    <manifest-hash>
      <hash xmlns='urn:xmpp:hashes:2' algo='sha-
256'>K2/YdwvIBAlfUIeZHiNCyiYA4m4W2wrEOqXQDoQyXp4=</hash>
    </manifest-hash>
  </task-state>
</iq>
```

Example 24. Caller retrieves the retained terminal result

```
<iq from='planner@agents.example/phone'
    id='task-result-1'
    to='weather@agents.example'
    type='get'>
  <task-result-request xmlns='urn:xmpp:agent-task:0'
                       task-id='task-7f2a9c41d8804e96bb478af6453c0371'/>
</iq>

<iq from='weather@agents.example'
    id='task-result-1'
    to='planner@agents.example/phone'
    type='result'>
  <task-result xmlns='urn:xmpp:agent-task:0'
               media-type='application/json'
               revision='4'
               state='completed'
               task-id='task-7f2a9c41d8804e96bb478af6453c0371'>
    {"result":{"content":[{"type":"text","text":"Light rain, 18 C"}],
               "structuredContent":{"summary":"Light rain, 18 C"},
```

```
                "isError":false}}
  </task-result>
</iq>
```

13.8 Message Routing, Storage, and Copies

Caller commands and recovery IQs MUST be addressed to the target bare JID. The authorization principal is the caller bare JID. Best-effort event messages SHOULD be addressed to the full JID that issued the <invoke/> element; after loss of that route the target MAY address notifications to the caller bare JID according to ordinary routing. Correctness MUST NOT depend on notification delivery to either route.

Because bare-JID notification fallback is permitted, this profile MUST NOT use Message Processing Hints (XEP-0334) [14] `no-copy`. A target MAY use `no-store` or `no-permanent-store` as a storage preference, but processing hints are never a correctness or confidentiality mechanism. Implementers MUST assume that carbons, archives, offline storage, and administrative logging can replicate task payloads. Events are deduplicated by `(target bare JID, task-id, revision)`, and an identical replay retains its event ID. Sensitive arguments or results SHOULD use an appropriate end-to-end encryption profile.

13.9 Idempotency and Delegation

The wire `request-id` and replay fingerprint provide invocation idempotency. They are scoped to the authenticated caller bare JID and target bare JID and are not authentication secrets.

The base protocol authenticates only the immediate stanza sender. It defines no signed delegation token, original-user assertion, approval grant, or chain-of-custody proof. A target MUST authorize the immediate authenticated sender. A local deployment MAY additionally enforce an original principal learned through trusted local context, but MUST NOT present that value to a remote peer as verifiable authority. Cross-domain on-behalf-of authorization is future work.

14. Human Messaging Profile

This optional application profile remains in this document but is not part of Core Directory, Tool Discovery, or Task Execution conformance. An agent endpoint MAY participate in ordinary direct chats independently of tool execution. Groupchat participation requires the endpoint to implement and advertise the applicable Multi-User Chat (XEP-0045) [15] behavior and is otherwise out of scope.

A human or channel conversation is channel-owned: normal message routing selects the conversation session, and normal channel-delivery rules own the response. It MUST NOT be treated as a task merely because an agent or LLM produces the reply.

A tool invocation remains task-owned even when its structured result contains natural-language text. It returns through the correlated task lifecycle and MUST NOT inherit a human channel's conversation session or delivery route merely because the result is prose.

An endpoint participating in human messaging MUST advertise only the extensions it implements. The deployed profile uses:

- Chat State Notifications (XEP-0085) [16] for chat state notifications;
- Message Delivery Receipts (XEP-0184) [5] for explicitly requested receipts on one-to-one chat messages, but not groupchat or task messages;
- Message Processing Hints (XEP-0334) [14] `store` hints for one-to-one messages intended for offline delivery;
- Unique and Stable Stanza IDs (XEP-0359) [17] origin IDs for stable message correlation;
- Message Replies (XEP-0461) [18] for replies;
- Data Forms (XEP-0004) [10] for structured questions requiring a human response; and
- standard XMPP presence subscription and probe behavior for virtual endpoint JIDs.

A gateway MUST keep human conversations separate from task execution state and MUST keep task correlation separate from channel destinations. Application output cannot choose its transport path.

An application MAY persist an authenticated requester's reply route with a durable task and, after completion, deliver the target's response from the target endpoint JID. In such a flow the calling agent supplies an instruction, not text authored on the target's behalf; the target authors its own response; the runtime delivers that response without asking the caller to repeat it; and the caller receives only delivery status and can acknowledge naturally. This application behavior does not change the base task result route and MUST obey recipient consent, blocking, provenance, and deduplication policy.

15. Error Handling

An IQ of type "get" or "set" that is not handled MUST receive an IQ error as required by XMPP Core. The original request payload MAY be included only when it is safe and within the stanza-size limit; a responder SHOULD omit or truncate attacker-controlled JSON to prevent reflection amplification and secret disclosure.

Table 10: Recommended stanza errors

Condition	Use
<bad-request/>	Malformed JSON, missing attributes, invalid tool name, or invalid endpoint identifier.
<forbidden/>	Registration for another JID or a denied operation where existence disclosure is acceptable.
<item-not-found/>	Unknown, hidden, expired, or unauthorized endpoint, tool, version, schema, task, or input request.
<not-acceptable/>	Unsupported manifest version, schema mismatch, failed JSON Schema validation, or expired deadline.
<conflict/>	Conflicting registration or conflicting terminal task event.
<resource-constraint/>	Manifest or schema too large, too many pending IQs, task capacity exhausted, or rate limit reached.
<service-unavailable/>	The addressed protocol, endpoint runtime, or optional feature is unavailable.

To avoid cross-principal enumeration, a gateway MUST return an <item-not-found/> error for an endpoint or task that exists but is not visible or authorized to the requester, using the same safe error payload as a nonexistent object.

Execution failures after task acceptance MUST use a `failed` lifecycle event with a stable application error code, a safe human-readable message, and a `retryable` boolean. Error messages MUST NOT expose secrets, internal prompts, stack traces, or data belonging to another principal.

Table 11: Reserved task failure codes

Code	Meaning
`deadline-expired`	The deadline elapsed before a successful terminal result.
`internal-error`	The target failed because of an unexpected internal condition.
`unavailable`	A required runtime or downstream service is temporarily unavailable.

These codes are reserved by this specification; no new central registry is created. A deployment-defined code MUST be namespace-qualified, for example `com.example.weather:quota-exceeded`. Pre-admission argument, permission, version, manifest-hash, and replay failures use IQ errors and are not task failure codes.

16. Business Rules

- A self-registered manifest JID MUST match the authenticated C2S principal; virtual endpoints require trusted administrative provisioning.
- An endpoint API version is immutable, and every tool list, tool query, schema request, invocation, task state, and result MUST remain pinned to its exact manifest.
- Directory visibility and invocation permission are separate decisions. Visible does not imply invocable.
- Search, directory items, endpoint information, tool information, and schema retrieval MUST each apply the relevant visibility policy.
- A caller gateway MAY prevalidate invocation arguments; the target MUST validate them against the authoritative registered schema. A caller SHOULD validate structured results on return.
- The target MUST be the only principal allowed to report progress or settle its task.
- An authorized resource of the immediate caller bare JID MUST be the only principal allowed to answer requested input, request cancellation, or recover state and results.
- Terminal states are immutable, except that an identical repeated terminal event MAY be treated as an idempotent replay.
- Approval-required or permission-bearing tools MUST fail closed until the gateway has an enforceable grant.
- Endpoint and tool annotations MUST be treated as untrusted metadata.
- A gateway MUST bound manifest size, schema size, result size, pending IQ count, task count, task duration, task-state retention, directory result count, and delegation depth.

17. Implementation Appendix

This section is informative. It records implementation guidance, MCP mapping, caching behavior, and deployed predecessor status separately from the normative protocol definition.

17.1 Gateway and Model Boundary

This subsection is informative. The XMPP gateway is the protocol endpoint; an LLM, provider runtime, or tool implementation need not parse or generate XMPP stanzas, JIDs, namespace identifiers, tenant identifiers, or delivery envelopes. A useful implementation presents a transport-neutral contract

containing the selected tool name, validated arguments, safe tool metadata, and only the event data needed to perform the work. Opaque endpoint and task handles reduce accidental transport disclosure.

After wire validation, a target gateway can dispatch a task to an internal agent or model runtime. Task-scoped provider state should remain separate from ordinary channel conversation state. An action runtime can report completion, failure, progress, input requests, or cancellation through a transport-neutral action bound to the current task; the gateway then applies the state transition and constructs the authorized stanza.

These architectural techniques do not replace the authorization invariants in Security Considerations. In particular, application output cannot select a wire sender, authenticated caller, tenant, unrelated task, or unauthorized destination.

17.2 MCP Mapping

Table 12: MCP to XMPP agent protocol mapping

MCP concept	XMPP representation
Server identity	Endpoint bare JID.
Implementation name and version	Distinct `implementation` manifest object; not the endpoint API version.
MCP protocol revision	Optional `mcpProtocolVersion`; not the manifest model version.
`tools/list`	Version-pinned `disco#items` tool collection followed by tool information queries.
`Tool`	Tool item, tool information form, and separately retrievable schemas.
`Tool.execution`	Preserved with MCP `taskSupport` semantics; XMPP hints use the `urn:xmpp:agent-api:0` extension object.
`tools/call`	IQ containing an <invoke/> element, asynchronous notifications, and state/result recovery.
`CallToolResult`	The complete object is preserved as terminal `result`.

`CallToolResult.isError=true`	A completed tool result, not an XMPP task-infrastructure failure.
`_meta` and annotations	Preserved where representable; unknown metadata MUST NOT affect authorization.
Progress notification	`progress` event.
Elicitation or human input	`input_required` notification and IQ containing a <provide-input/> element; this is not a transparent MCP elicitation tunnel.
Authorization	XMPP-authenticated principals plus gateway policy; HTTP OAuth tokens are not forwarded.

MCP clients connected through a bridge SHOULD receive normal MCP tool objects. A bridge MUST preserve schemas, annotations, `execution`, `content`, `structuredContent`, `isError`, and `_meta` where representable; preserve MCP tool-list cursors internally; and report semantic loss when either side cannot represent a capability. The XMPP directory RSM cursor is not an MCP tool-list cursor.

MCP 2025-11-25 Tasks are experimental and not wire-identical to this protocol. Several detailed XMPP non-terminal states map to MCP `working`; MCP `working` cannot be expanded back into a more specific XMPP state without bridge-owned knowledge. MCP TTL is measured in milliseconds from task creation, while this protocol advertises an absolute retention deadline covering the task result. Receiver-generated task IDs, creation responses, polling, result retrieval, and cancellation MUST be represented explicitly in a bridge-loss table. A bridge MUST NOT claim transparent MCP Tasks support when any of these semantics are lost.

17.3 Caching and Change Notification

Clients MAY cache manifests, endpoint information, tool information, and schemas by endpoint bare JID, exact API version, Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] hash, authenticated principal, and a trusted local visibility-scope identifier. Cache entries MUST NOT cross principals or authorization scopes unless policy proves the returned view is identical. Because manifests are immutable and all-or-nothing, a matching version and hash identifies the same registered content. A future extension can define push notification for active-version changes; the base protocol requires clients to rediscover.

17.4 Canonical JSON

The standards profile uses RFC 8785 canonicalization and Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] hashes. The deployed private profile used a simpler key-sorting algorithm; its values are comparable only within that profile. A client that cannot reproduce the declared canonicalization MAY treat a hash as an opaque cache validator but MUST still validate received JSON and schema content.

17.5 Implementation Status and Known Differences

The implementation on which this draft is based has been integration-tested using two independently configured Openfire server instances connected by XMPP server-to-server federation, with the gateway hosted on one domain and reached from the other. The tests exercise deny-by-default federation and independently configured visibility and invocation grants. This is evidence from one gateway implementation in an Openfire-based multi-domain deployment, not a claim of interoperability between two independent implementations of this protocol.

Table 13: Implementation and protocol status

Capability	NanoClaw implementation of the :0 protocol	Independent interoperability test
External-component hosting and endpoint discovery	Implemented	No
Visibility-scoped directory and search with RSM	Implemented	No
Immutable manifest retrieval and explicit activation	Implemented	No
Version-pinned schemas and tools	Implemented	No
IQ durable task admission and idempotent request replay	Implemented	No
Progress, input, cancellation, and settlement	Implemented	No
Complete state and result recovery within the promised retention period	Implemented	No

Required JSON Schema 2020-12 evaluation	Implemented with bounded execution	No
Federated visibility and invocation policy	Deny by default; independently configurable allowlists	No
Human messaging and requester delivery	Implemented as a non-conformance application profile	No

The deployed runtime enforces the gateway/model boundary described above. Model-facing discovery returns opaque `agent:` handles rather than XMPP endpoint identifiers. Inbound agent tasks are formatted as transport-neutral tool, argument, and event data; wire JIDs, component domains, tenant and workspace routing, endpoint addresses, and stanza correlation remain host-owned. Action tasks run in task-scoped sessions and settle only through explicit task actions; provider final prose cannot escape through a human channel.

The deployed application defines a local tool named `conversation.respond`. This is an application convention, not a well-known name or interoperable feature defined by this specification. It uses the durable task transport, while the runtime converts the target's non-empty prose into the application's declared structured result. Implementations using that name remain responsible for publishing and versioning their own schemas and result-size limits.

The deployed caller API additionally supports a local `delivery="requester"` policy for that application tool. It resolves the active XMPP requester route from session state, persists it with the task, sends the completed response through the channel adapter with the target endpoint JID as sender identity, records delivery state, and returns that outcome to the calling agent. This policy has no representation or negotiation in the protocol defined by this document and is therefore not part of conformance.

The deployed direct-delivery path currently sends an ordinary chat message without a standardized indication that another agent initiated the interaction. It also has at-least-once semantics: a crash after network send but before committing the delivery marker can expose the same response to the human twice. A future interoperable delivery profile needs explicit provenance, recipient-consent and blocking behavior, and end-to-end acknowledgement or deduplication.

The version 0 implementation replaces the predecessor wire profile rather than operating as a dual stack. It uses the fixed `'urn:xmpp:*:0'` namespaces, immutable manifests with explicit activation, standard JSON media types, RFC 8785 canonicalization, Use of Cryptographic Hash Functions in XMPP (XEP-0300) [9] hashes, version-pinned tool nodes and schemas, receiver-generated task IDs, revisioned

lifecycle events, complete recovery IQs, and authorization derived from the authenticated XMPP sender and trusted local policy. Directory, search, and tool paging use Result Set Management (XEP-0059) [2]. The normative blocks in this document are intended to be the source for committed manifest, event, and XML Schema artifacts. Implementations SHOULD normalize and compare generated artifacts byte-for-byte and check protocol namespace constants separately.

The following work remains outside the implemented conformance surface:

- permission-bearing and approval-required tools fail closed unless trusted local policy establishes the required authorization or approval;
- portable approval and delegation grants, registration expiry, and tool-list change notification remain future extensions; and
- the Openfire suites provide integration evidence only; no independent implementation interoperability result is claimed.

The predecessor implementation and its private identifiers are not a supported compatibility mode and are not part of conformance to this specification.

18. Future Extensions

The following work is intentionally left for later namespace versions or separate XEPs:

- verifiable original-principal delegation and parent-task ancestry on the wire;
- manifest expiry, lease renewal, and signed manifests;
- push notification when manifests or tool lists change;
- standardized authorization scopes, signed approval grants, and delegation tokens;
- discovery and transport for MCP resources, prompts, sampling, roots, and logging;
- binary and file inputs using existing XMPP file-sharing protocols;
- end-to-end encryption and signing profiles for task payloads;
- federated gateway-to-gateway trust and policy discovery;
- result streaming and resumable event history.

19. Accessibility Considerations

Agent, tool, question, progress, error, and approval interfaces SHOULD expose text labels to assistive technology and SHOULD NOT rely solely on color, icons, animation, or presence state. Clients SHOULD show the endpoint JID in addition to a display title so that users can distinguish look-alike agents.

Interactive input forms SHOULD use specific field types, labels, descriptions, validation errors, and deterministic focus order. Progress percentages SHOULD be accompanied by textual stages when available. Clients SHOULD allow users to review full tool arguments and the declared side-effect hints before approval.

20. Internationalization Considerations

JIDs MUST be prepared and compared according to the applicable XMPP address rules. Human-readable titles, descriptions, instructions, questions, summaries, and errors SHOULD carry or inherit the 'xml:lang' attribute where they are represented as XML text. JSON manifest text is not language-tagged in this version; implementations that localize it SHOULD select a locale through deployment policy and MUST retain stable non-localized names for matching and invocation.

Search matching SHOULD use locale-appropriate Unicode case handling. Implementations MUST NOT use locale-dependent matching for authorization, JID comparison, tool names, versions, hashes, request IDs, or task IDs.

21. Security Considerations

Tool invocation can produce arbitrary external side effects. Implementations MUST authenticate the XMPP sender, authorize every tool call, validate all inputs, and fail closed. Discovery metadata and MCP annotations are not authority. Destructive tools SHOULD require an explicit grant or human approval appropriate to the deployment.

A gateway MUST prevent sender spoofing when acting for virtual endpoint JIDs. Self-registration MUST prove control of the endpoint JID; administrative provisioning MUST authenticate and authorize the administrator. Tenant and workspace identifiers are deployment-local context and are not base-protocol identity assertions.

JSON parsing, JSON Schema evaluation, regular-expression evaluation, and XML parsing are attacker-controlled surfaces. Implementations MUST bound nesting, size, execution time, pending work, identifier length, and schema complexity. External `$ref` resolution MUST be disabled by default and MAY be enabled only with an explicit allowlist and resource budget. XML parsers MUST disable unsafe external entity resolution for protocol payloads.

Schema hashes provide change detection, not signer authentication. A hash received from the same untrusted source as the schema does not make the schema trustworthy. Signed manifests and a trust model are future work.

Request IDs, task IDs, and event IDs MUST satisfy the identifier grammar and 128-bit unpredictability requirement in Invocation. Knowledge of an identifier MUST NOT grant access. Task lookup, cancellation, input, and results MUST all authorize the immediate caller bare JID. Unknown, expired, and unauthorized lookups MUST be indistinguishable.

Gateways MUST rate-limit registration, directory enumeration, search, schema retrieval, invocation, progress events, input requests, and result sizes. Cold-start support is especially susceptible to resource exhaustion and SHOULD have quotas and concurrency limits.

Task retries are dangerous. Stream loss, IQ timeout, and missing progress do not prove non-execution. A caller MUST retry the identical request ID and fingerprint or use task-specific reconciliation; it MUST NOT create a fresh side-effecting request merely because admission was not observed.

Component secrets, OAuth tokens, API keys, cookies, and other credentials MUST NOT appear in manifests, discovery forms, ordinary chat bodies, task context, progress, or error text. A gateway bridging to MCP over HTTP MUST NOT pass through a caller token to a downstream service unless an explicit secure token-exchange design permits it.

Tool descriptions, schemas, arguments, results, and input questions are untrusted content and can contain prompt-injection instructions. A gateway or agent MUST NOT treat such content as policy, authorization, or operator instruction. Tool output MUST remain data until an explicitly authorized application interprets it.

Version 0 defines no portable approval token. A future approval profile must cryptographically bind a grant to the caller, endpoint, tool, immutable manifest, exact canonical arguments or an explicitly bounded argument scope, expiry, nonce or request ID, and intended side effect. Interactive input alone is not such a grant.

An LLM is not an XMPP security principal. Hiding transport identifiers from the model reduces accidental disclosure but does not replace authorization. The gateway MUST bind every model-facing opaque endpoint handle and task handle to server-side identity and tenant state, reject stale or cross-context handles, and construct all wire addresses itself. Model-generated destination markup or identity claims MUST be ignored for task delivery.

A server or component MUST originate a stanza from a virtual target JID only when it is authorized to act for that endpoint. A caller gateway MUST NOT spoof a remote target JID; cross-domain sending from the target requires the target gateway or a future explicit delegation protocol.

An implementation that performs application-specific direct delivery of a task result to a human MUST establish recipient consent, either explicitly or through a documented closed-deployment policy, and

MUST honor Blocking Command (XEP-0191) [19] blocking and applicable presence or messaging policy. It SHOULD attach provenance with Message Replies (XEP-0461) [18] when the triggering message is known so a client can render the relationship to the initiating conversation. Without an end-to-end acknowledgement or deduplication profile, such human-visible delivery has at-least-once semantics: a crash after network send but before local commit can expose the same response twice.

Gateways are confused-deputy targets. The base protocol authorizes the immediate authenticated XMPP sender for the selected endpoint, exact tool, immutable manifest, canonical arguments, and intended side effect. It cannot prove an original principal across an agent chain. A local original-user identity, delegation ancestry, trusted agent name, or prior approval for another call MUST NOT confer remote authority without a future verifiable delegation or approval grant.

Federated discovery, registration, and invocation MUST be denied by default unless an operator explicitly enables the remote domain or principal. Allowlists SHOULD be scoped separately for visibility and invocation. Display names and localparts are subject to name squatting and visual impersonation; clients SHOULD emphasize the normalized endpoint JID and security context.

22. Privacy Considerations

Directory listing and search can reveal the existence, names, functions, vendors, versions, and availability of agents. Tool descriptions and schemas can reveal internal business processes. Gateways MUST apply an explicit visibility policy and SHOULD reveal the minimum data necessary to unauthorized or anonymous requesters.

Task arguments, results, progress, input requests, caller JIDs, workspace identifiers, and task ancestry can contain personal or confidential data. Implementations SHOULD define retention limits, access logging, deletion behavior, and encryption appropriate to their threat model. Operator logs MUST redact secrets and SHOULD avoid full task payloads.

An application that persists an out-of-band conversation route with a task links the requester, caller agent, target agent, channel, and thread. Such a routing record SHOULD contain only the fields needed for delivery, MUST receive the same access control as the task, and SHOULD be deleted or minimized according to the task-retention policy.

Public profile data, tenant-scoped API metadata, and invocation authorization are distinct policy surfaces. Publishing a vCard or directory item MUST NOT make private tools or schemas public and MUST NOT imply invocation permission.

Task notification messages can be copied by Message Carbons (XEP-0280) [20], archived by Message Archive Management (XEP-0313) [21], retained in offline storage, and exposed to server operators in plaintext unless an end-to-end encryption profile is used. Task IQ payloads are likewise visible to routing servers and administrative logging. Deployments MUST document those replication and retention paths. Responses for hidden and nonexistent endpoints SHOULD be similar in timing, and equivalent rate limits SHOULD apply, to reduce directory-enumeration side channels.

23. IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA).

24. XMPP Registrar Considerations

24.1 Protocol Namespaces

The XMPP Registrar shall register the XML protocol namespaces `'urn:xmpp:agent-api:0'` and `'urn:xmpp:agent-task:0'`. FORM_TYPE values, feature vars, and discovery nodes are not XML protocol namespaces.

24.2 Service Discovery Identities

The XMPP Registrar shall add the following types to the `automation` category:

Table 14: Service discovery identity registrations

Type	Description
`agent-gateway`	A service that hosts, discovers, and routes for addressable AI agents.
`agent-directory`	A visibility-scoped directory of addressable agent endpoints.
`agent-endpoint`	An addressable agent endpoint exposing callable tools.
`agent-tool-collection`	A version-pinned collection of tools exposed by an agent endpoint.
`agent-tool`	A discovery node representing one callable agent tool.

24.3 Service Discovery Features

The XMPP Registrar shall register the fixed feature vars `urn:xmpp:agent-directory:0`, `urn:xmpp:agent-endpoint:0`, `urn:xmpp:agent-tool:0`, `urn:xmpp:agent-api:0#manifest`, `urn:xmpp:agent-api:0#schema`, `urn:xmpp:agent-api:0#self-register`, `urn:xmpp:agent-api:0#admin`, `urn:xmpp:agent-task:0`, `urn:xmpp:agent-task:0#progress`, `urn:xmpp:agent-task:0#cancel`, and `urn:xmpp:agent-task:0#input`, with the semantics defined by this document.

24.4 Data Form Types

The XMPP Registrar shall register the following field standardizations in accordance with Field Standardization for Data Forms (XEP-0068) [22]:

Endpoint information form registration

```
<form_type>
  <name>urn:xmpp:agent-endpoint-info:0</name>
  <doc>XEP-XXXX</doc>
  <desc>Information about an XMPP agent endpoint.</desc>
  <field label='Stable endpoint name' type='text-single' var='server_name'/>
  <field label='Display title' type='text-single' var='server_title'/>
  <field label='Description' type='text-multi' var='description'/>
  <field label='Active API version' type='text-single' var='version'/>
  <field label='Manifest hash algorithm' type='text-single'
var='manifest_hash_algo'/>
  <field label='Base64 manifest hash' type='text-single'
var='manifest_hash_value'/>
  <field label='Cold start supported' type='boolean'
var='cold_start_supported'/>
  <field label='Minimum request replay window in seconds' type='text-single'
var='request_replay_seconds'/>
</form_type>
```

Tool information form registration

```
<form_type>
  <name>urn:xmpp:agent-tool-info:0</name>
  <doc>XEP-XXXX</doc>
  <desc>Information about one version-pinned XMPP agent tool.</desc>
  <field label='Tool name' type='text-single' var='name'/>
  <field label='Display title' type='text-single' var='title'/>
  <field label='Description' type='text-multi' var='description'/>
  <field label='Endpoint API version' type='text-single' var='api_version'/>
  <field label='Input schema hash algorithm' type='text-single'
```

```
var='input_schema_hash_algo'/>
  <field label='Base64 input schema hash' type='text-single'
var='input_schema_hash_value'/>
  <field label='Output schema hash algorithm' type='text-single'
var='output_schema_hash_algo'/>
  <field label='Base64 output schema hash' type='text-single'
var='output_schema_hash_value'/>
  <field label='MCP read-only hint' type='boolean' var='read_only'/>
  <field label='MCP destructive hint' type='boolean' var='destructive'/>
  <field label='MCP idempotent hint' type='boolean' var='idempotent'/>
  <field label='MCP open-world hint' type='boolean' var='open_world'/>
</form_type>
```

24.5 Service Discovery Nodes

The XMPP Registrar shall register fixed node `urn:xmpp:agent-directory:0` and derived-node bases `urn:xmpp:agent-tools:0` and `urn:xmpp:agent-tool:0`. Derived tool collection nodes and tool nodes follow the exact grammar in Protocol Namespaces and Identifiers; they are discovery node identifiers, not XML namespaces.

24.6 Task Failure Codes

This document creates no failure-code registry. It reserves the unqualified codes listed in Error Handling. Extensions use namespace-qualified codes, avoiding a central collision registry.

25. XML Schema

The following schemas are normative for the base XML structures. Implementations MUST also enforce the state, hashing, JSON validation, extension, and authorization rules that XML Schema cannot express.

25.1 Agent API Schema

Schema for urn:xmpp:agent-api:0

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns='urn:xmpp:agent-api:0'
           xmlns:xs='http://www.w3.org/2001/XMLSchema'
           elementFormDefault='qualified'
           targetNamespace='urn:xmpp:agent-api:0'>
```

```
  <xs:simpleType name='version'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='[A-Za-z0-9._~-]{1,64}'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='toolName'>
    <xs:restriction base='xs:string'>
      <xs:minLength value='1'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='direction'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='input'/>
      <xs:enumeration value='output'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name='hashContainer'>
    <xs:sequence>
      <xs:any maxOccurs='1'
              minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name='register'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='manifest'>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base='xs:string'>
                <xs:attribute fixed='application/json' name='media-type'
                              type='xs:string' use='required'/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='registered'>
    <xs:complexType>
      <xs:sequence>
        <xs:any maxOccurs='1'
                minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
      </xs:sequence>
      <xs:attribute name='jid' type='xs:string' use='required'/>
      <xs:attribute name='version' type='version' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='activate'>
    <xs:complexType>
      <xs:sequence>
        <xs:any maxOccurs='1'
                minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
      </xs:sequence>
      <xs:attribute name='jid' type='xs:string' use='required'/>
      <xs:attribute name='version' type='version' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='active'>
    <xs:complexType>
      <xs:attribute name='jid' type='xs:string' use='required'/>
      <xs:attribute name='version' type='version' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name='endpointOperation'>
    <xs:attribute name='jid' type='xs:string' use='required'/>
  </xs:complexType>

  <xs:element name='deactivate' type='endpointOperation'/>
```

```
  <xs:element name='deactivated' type='endpointOperation'/>
  <xs:element name='registration-status' type='endpointOperation'/>
  <xs:element name='remove-endpoint' type='endpointOperation'/>

  <xs:element name='remove-version'>
    <xs:complexType>
      <xs:attribute name='jid' type='xs:string' use='required'/>
      <xs:attribute name='version' type='version' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='removed'>
    <xs:complexType>
      <xs:attribute name='jid' type='xs:string' use='required'/>
      <xs:attribute name='version' type='version' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='registration'>
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs='unbounded' minOccurs='0' name='version'>
          <xs:complexType>
            <xs:sequence>
              <xs:any maxOccurs='1'
                      minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
            </xs:sequence>
            <xs:attribute name='id' type='version' use='required'/>
            <xs:attribute name='active' type='xs:boolean' use='required'/>
            <xs:attribute name='removable' type='xs:boolean' use='required'/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name='jid' type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='manifest-request'>
    <xs:complexType>
```

```
      <xs:attribute name='version' type='version' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='manifest'>
    <xs:complexType>
      <xs:sequence>
        <xs:any maxOccurs='1'
                minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
        <xs:element name='json' type='xs:string'/>
      </xs:sequence>
      <xs:attribute name='version' type='version' use='required'/>
      <xs:attribute fixed='application/json' name='media-type'
                    type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='schema-request'>
    <xs:complexType>
      <xs:sequence>
        <xs:any maxOccurs='1'
                minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
      </xs:sequence>
      <xs:attribute name='tool' type='toolName' use='required'/>
      <xs:attribute name='version' type='version' use='required'/>
      <xs:attribute name='direction' type='direction' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='schema'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='manifest-hash' type='hashContainer'/>
        <xs:element name='schema-hash' type='hashContainer'/>
        <xs:element name='json' type='xs:string'/>
      </xs:sequence>
      <xs:attribute name='tool' type='toolName' use='required'/>
      <xs:attribute name='version' type='version' use='required'/>
```

```
      <xs:attribute name='direction' type='direction' use='required'/>
      <xs:attribute fixed='application/schema+json' name='media-type'
                    type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

25.2 Agent Task Schema

Schema for urn:xmpp:agent-task:0

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns='urn:xmpp:agent-task:0'
           xmlns:xs='http://www.w3.org/2001/XMLSchema'
           elementFormDefault='qualified'
           targetNamespace='urn:xmpp:agent-task:0'>

  <xs:simpleType name='eventType'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='status'/>
      <xs:enumeration value='progress'/>
      <xs:enumeration value='input_required'/>
      <xs:enumeration value='completed'/>
      <xs:enumeration value='failed'/>
      <xs:enumeration value='cancelled'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='executionState'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='accepted'/>
      <xs:enumeration value='running'/>
      <xs:enumeration value='input_required'/>
      <xs:enumeration value='cancelling'/>
      <xs:enumeration value='cancelled'/>
      <xs:enumeration value='failed'/>
      <xs:enumeration value='completed'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='terminalState'>
    <xs:restriction base='xs:string'>
```

```
      <xs:enumeration value='cancelled'/>
      <xs:enumeration value='failed'/>
      <xs:enumeration value='completed'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='identifier'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='[A-Za-z0-9._~-]{22,128}'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='version'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='[A-Za-z0-9._~-]{1,64}'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='toolName'>
    <xs:restriction base='xs:string'>
      <xs:minLength value='1'/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name='hashContainer'>
    <xs:sequence>
      <xs:any maxOccurs='1'
              minOccurs='1' namespace='urn:xmpp:hashes:2'
processContents='lax'/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name='invoke'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='manifest-hash' type='hashContainer'/>
        <xs:element name='arguments'>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base='xs:string'>
```

```
              <xs:attribute fixed='application/json' name='media-type'
                            type='xs:string' use='required'/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element minOccurs='0' name='deadline' type='xs:dateTime'/>
    </xs:sequence>
    <xs:attribute name='request-id' type='identifier' use='required'/>
    <xs:attribute name='tool' type='toolName' use='required'/>
    <xs:attribute name='api-version' type='version' use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='accepted'>
  <xs:complexType>
    <xs:attribute name='request-id' type='identifier' use='required'/>
    <xs:attribute name='task-id' type='identifier' use='required'/>
    <xs:attribute name='revision' type='xs:nonNegativeInteger'
use='required'/>
    <xs:attribute name='created' type='xs:dateTime' use='required'/>
    <xs:attribute name='retain-until' type='xs:dateTime' use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='provide-input'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='input'>
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base='xs:string'>
              <xs:attribute fixed='application/json' name='media-type'
                            type='xs:string' use='required'/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name='task-id' type='identifier' use='required'/>
```

```
      <xs:attribute name='request-id' type='identifier' use='required'/>
      <xs:attribute name='expected-revision' type='xs:nonNegativeInteger'
use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='input-accepted'>
    <xs:complexType>
      <xs:attribute name='task-id' type='identifier' use='required'/>
      <xs:attribute name='revision' type='xs:nonNegativeInteger'
use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='cancel'>
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs='0' name='reason' type='xs:string'/>
      </xs:sequence>
      <xs:attribute name='task-id' type='identifier' use='required'/>
      <xs:attribute name='expected-revision' type='xs:nonNegativeInteger'
use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='cancel-accepted'>
    <xs:complexType>
      <xs:attribute name='task-id' type='identifier' use='required'/>
      <xs:attribute name='revision' type='xs:nonNegativeInteger'
use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='event'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='xs:string'>
          <xs:attribute name='task-id' type='identifier' use='required'/>
          <xs:attribute name='event-id' type='identifier' use='required'/>
          <xs:attribute name='revision' type='xs:nonNegativeInteger'
```

```
use='required'/>
          <xs:attribute name='type' type='eventType' use='required'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='task-state-request'>
    <xs:complexType>
      <xs:attribute name='task-id' type='identifier' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='task-state'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='manifest-hash' type='hashContainer'/>
        <xs:element minOccurs='0' name='pending-input'>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base='xs:string'>
                <xs:attribute fixed='application/json' name='media-type'
                              type='xs:string' use='required'/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name='task-id' type='identifier' use='required'/>
      <xs:attribute name='endpoint' type='xs:string' use='required'/>
      <xs:attribute name='state' type='executionState' use='required'/>
      <xs:attribute name='revision' type='xs:nonNegativeInteger'
use='required'/>
      <xs:attribute name='api-version' type='version' use='required'/>
      <xs:attribute name='created' type='xs:dateTime' use='required'/>
      <xs:attribute name='updated' type='xs:dateTime' use='required'/>
      <xs:attribute name='deadline' type='xs:dateTime' use='optional'/>
      <xs:attribute name='retain-until' type='xs:dateTime' use='required'/>
      <xs:attribute name='result-available' type='xs:boolean'
use='required'/>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name='task-result-request'>
    <xs:complexType>
      <xs:attribute name='task-id' type='identifier' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='task-result'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='xs:string'>
          <xs:attribute name='task-id' type='identifier' use='required'/>
          <xs:attribute name='state' type='terminalState' use='required'/>
          <xs:attribute name='revision' type='xs:nonNegativeInteger'
use='required'/>
          <xs:attribute fixed='application/json' name='media-type'
                        type='xs:string' use='required'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

26. Acknowledgements

This protocol grew out of real-world work on agent orchestration, including implementation and integration testing with Openfire, and was refined through multiple independent technical reviews. It also builds on the tool model developed by the Model Context Protocol community and on the XMPP specifications cited throughout this document.

Appendices

Appendix A: Document Information

Series

XEP

Number

xxxx

Publisher

XMPP Standards Foundation

Status

ProtoXEP

Type

Standards Track

Version

0.0.1

Last Updated

2026-07-28

Approving Body

XMPP Council

Dependencies

XMPP Core, XMPP IM, RFC 4648, RFC 8785, JSON Schema 2020-12, Model Context Protocol, XEP-0004, XEP-0030, XEP-0055, XEP-0059, XEP-0068, XEP-0082, XEP-0114, XEP-0128, XEP-0191, XEP-0199, XEP-0300, XEP-0334, XEP-0461

Supersedes

None

Superseded By

None

Short Name

NOT_YET_ASSIGNED

This document in other formats: XML PDF

Appendix B: Author Information

Roman Shterenzon

Email
roman.shterenzon@gmail.com

JabberID
roman.shterenzon@gmail.com

Appendix C: Legal Notices

Copyright

Permissions

Disclaimer of Warranty

Limitation of Liability

shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

IPR Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Visual Presentation

The HTML representation (you are looking at) is maintained by the XSF. It is based on the YAML CSS Framework, which is licensed under the terms of the CC-BY-SA 2.0 license.

Appendix D: Relation to XMPP

The Extensible Messaging and Presence Protocol (XMPP) is defined in the XMPP Core (RFC 6120) and XMPP IM (RFC 6121) specifications contributed by the XMPP Standards Foundation to the Internet Standards Process, which is managed by the Internet Engineering Task Force in accordance with RFC 2026. Any protocol defined in this document has been developed outside the Internet Standards Process and is to be understood as an extension to XMPP rather than as an evolution, development, or modification of XMPP itself.

Appendix E: Discussion Venue

The primary venue for discussion of XMPP Extension Protocols is the <standards@xmpp.org> discussion list.

Discussion on other xmpp.org discussion lists might also be appropriate; see <https://xmpp.org/community/> for a complete list.

Given that this XMPP Extension Protocol normatively references IETF technologies, discussion on the <xsf-ietf@xmpp.org> list might also be appropriate.

Errata can be sent to <editor@xmpp.org>.

Appendix F: Requirements Conformance

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

Appendix G: Notes

1. XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

2. XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.

3. XEP-0128: Service Discovery Extensions <https://xmpp.org/extensions/xep-0128.html>.

4. XEP-0055: Jabber Search <https://xmpp.org/extensions/xep-0055.html>.

5. XEP-0184: Message Delivery Receipts <https://xmpp.org/extensions/xep-0184.html>.

6. XEP-0335: JSON Containers <https://xmpp.org/extensions/xep-0335.html>.

7. XEP-0114: Jabber Component Protocol <https://xmpp.org/extensions/xep-0114.html>.

8. XEP-0199: XMPP Ping <https://xmpp.org/extensions/xep-0199.html>.

9. XEP-0300: Use of Cryptographic Hash Functions in XMPP <https://xmpp.org/extensions/xep-0300.html>.

10. XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

11. XEP-0292: vCard4 over XMPP <https://xmpp.org/extensions/xep-0292.html>.

12. XEP-0054: vcard-temp <https://xmpp.org/extensions/xep-0054.html>.

13. XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.

14. XEP-0334: Message Processing Hints <https://xmpp.org/extensions/xep-0334.html>.

15. XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

16. XEP-0085: Chat State Notifications <https://xmpp.org/extensions/xep-0085.html>.

17. XEP-0359: Unique and Stable Stanza IDs <https://xmpp.org/extensions/xep-0359.html>.

18. XEP-0461: Message Replies <https://xmpp.org/extensions/xep-0461.html>.

19. XEP-0191: Blocking Command <https://xmpp.org/extensions/xep-0191.html>.

20. XEP-0280: Message Carbons <https://xmpp.org/extensions/xep-0280.html>.

21. XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.

22. XEP-0068: Field Data Standardization for Data Forms <https://xmpp.org/extensions/xep-0068.html>.

Appendix H: Revision History

Note: Older versions of this specification might be available at https://xmpp.org/extensions/attic/

1. Version 0.0.1 (2026-07-28)
   First draft.

   RS

Appendix I: Bib(La)TeX Entry

```
@report{shterenzon2026xepxxxx,
  title = {XMPP Agent Gateway},
  author = {Shterenzon, Roman},
  type = {XEP},
  number = {xxxx},
  version = {0.0.1},
  institution = {XMPP Standards Foundation},
  url = {https://xmpp.org/extensions/xep-xxxx.html},
  date = {2026-07-28/2026-07-28},
}
```

END