SPECIFICATION - 22 MAY 2026

# Agent Asynchronous Messaging Protocol (AAMP) Core Specification

Core control-plane semantics for interoperable asynchronous task exchange over Internet mail, with an optional streaming observation extension.

This version reflects the AAMP 1.1 implementation baseline.


## Abstract

This specification defines the Agent Asynchronous Messaging Protocol (AAMP), a mailbox-native protocol for asynchronous task dispatch, acknowledgement, clarification, cancellation, pairing authorization, and result delivery among independent participants. AAMP reuses ordinary email transport and message threading while adding a compact set of structured `X-AAMP-*` header fields for machine-readable semantics. The protocol is designed to support collaboration among agent runtimes, workflow systems, and human operators without requiring shared proprietary APIs.


## Status of This Document

This document defines the AAMP 1.1 core specification and its relationship to optional compatibility profiles for streaming and SDK-targeted deployments.

The core protocol described here is intentionally smaller than the full implementation surface of the current AAMP reference deployment. In particular, mailbox registration, credential exchange, inbox listing, thread retrieval, directory services, and other deployment-specific helper actions are out of scope for core conformance. The optional streaming observation extension is described as a separate conformance layer.

### Table of Contents

---

# 1. Conformance

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in RFC 2119 and RFC 8174.

## 1.1 Conformance Classes

An implementation conforms to this specification if it satisfies one of the following classes.

| Class | Requirements |
| --- | --- |
| Core AAMP Node | Implements the core lifecycle intents, the required header fields, the discovery document, and the processing requirements in Sections 4 through 8. |

| Class | Requirements |
| --- | --- |
| Stream-Capable AAMP Node | Implements the Core AAMP Node profile and the optional streaming observation extension defined in Section 9. |

---

## 2. Terminology

- **Node**: A participant with an AAMP-capable mailbox identity.
- **Dispatcher**: A node that sends `task.dispatch`.
- **Executor**: A node that receives and processes a task.
- **Task Thread**: The message thread associated with a single task identifier.
- **Control Plane**: The mail thread that carries authoritative lifecycle messages.
- **Data Plane**: The optional stream associated with a task thread.
- **Discovery Document**: The JSON representation published at `/.well-known/aamp`.
- **Sender Policy**: A receiver-local authorization rule that decides which senders may dispatch work, optionally scoped by dispatch-context rules.
- **Pairing Code**: A short-lived one-time code generated by a receiver and carried through an `aamp://connect` URL to authorize a new sender.

---

## 3. Protocol Overview

AAMP defines a small, structured task vocabulary on top of Internet mail. Rather than standardizing a new transport, it leverages the ubiquity, durability, and auditability of mail while constraining only those semantics needed for interoperable task execution.

The protocol standardizes task lifecycle intents (`task.dispatch`, `task.cancel`, `task.ack`, `task.help_needed`, and `task.result`), the optional streaming observation intent (`task.stream.opened`), pairing authorization intents (`pair.request` and `pair.respond`), and portable capability-card intents (`card.query` and `card.response`).

> The control plane is authoritative. The optional stream does not replace the final `task.result` message.

---

# 4. Transport and Message Model

## 4.1 Message Substrate

Each AAMP message is an Internet mail message. A typical deployment uses SMTP submission and relay for outbound delivery, RFC 5322 and MIME for message formatting, and JMAP for mailbox retrieval, synchronization, and attachment access. This specification does not require a single mail protocol stack so long as the required message fields are preserved.

## 4.2 Threading

AAMP uses the mailbox thread as the conversational container for one task identifier. Senders SHOULD generate a globally unique `Message-ID` and SHOULD preserve reply threading through `In-Reply-To` and `References` when replying within a task thread.

## 4.3 Header and Body Separation

AAMP reserves headers for machine-readable metadata. Human-readable instructions, output, blocked reasons, and narrative context SHOULD appear in the body. Attachments MAY be used for artifacts, generated files, or domain-specific payloads.

## 4.4 General Parsing Requirements

- Receivers MUST treat AAMP header field names as case-insensitive.
- Receivers MUST ignore unknown AAMP extensions unless they are explicitly required by a profile.
- Receivers SHOULD preserve unknown fields for forwarding, logging, or audit purposes.
- Receivers SHOULD de-duplicate repeated deliveries where possible.

---

# 5. Header Fields

## 5.1 Required Fields

| Field | Requirement | Description |
|---|---|---|
| `X-AAMP-Version` | Required on all AAMP messages | Protocol version string. This specification defines `1.1`. |
| `X-AAMP-Intent` | Required on all AAMP messages | Identifies the lifecycle intent of the message. |
| `X-AAMP-TaskId` | Required on all AAMP messages | Unique identifier for the task thread. |

## 5.2 Optional Fields

| Field | Used By | Description |
|---|---|---|
| `X-AAMP-Priority` | `task.dispatch` | Scheduling hint with values `urgent`, `high`, or `normal`. |
| `X-AAMP-Expires-At` | `task.dispatch` | Absolute ISO 8601 timestamp after which the task SHOULD be treated as stale. |
| `X-AAMP-Session-Key` | `task.dispatch` | Stable conversation or routing key for runtimes that should reuse an underlying agent session across multiple task turns. |
| `X-AAMP-Dispatch-Context` | `task.dispatch` | Percent-encoded semicolon-separated key-value pairs for portable routing or authorization context. |
| `X-AAMP-ParentTaskId` | `task.dispatch` | Optional parent task identifier for nested workflows. |
| `X-AAMP-Status` | `task.result`, `pair.respond` | Terminal status or pairing response status. This specification defines `completed` and `rejected`. |
| `X-AAMP-ErrorMsg` | `task.result`, `pair.respond` | Machine-readable or short human-readable rejection reason. |

| Field | Used By | Description |
|---|---|---|
| `X-AAMP-StructuredResult` | `task.result` | Base64url-encoded UTF-8 JSON for machine-readable result payloads defined by an application profile. |
| `X-AAMP-SuggestedOptions` | `task.help_needed` | Pipe-delimited suggested responses or next actions. |
| `X-AAMP-Stream-Id` | `task.stream.opened` | Identifier for the optional associated stream. |
| `X-AAMP-Pair-Code` | `pair.request` | One-time pairing code extracted from an `aamp://connect` URL. |
| `X-AAMP-Dispatch-Context-Rules` | `pair.request` | Base64url-encoded UTF-8 JSON object describing dispatch-context rules to attach to the newly paired sender policy. |
| `X-AAMP-Card-Summary` | `card.response` | One-line portable capability-card summary. |

### 5.3 Dispatch Context Encoding

Keys in `X-AAMP-Dispatch-Context` SHOULD consist only of lowercase ASCII letters, digits, underscore, and hyphen. Values SHOULD be percent-encoded UTF-8 text. Receivers MAY ignore invalid entries.

```
X-AAMP-Dispatch-Context: project_key=proj_123; user_key=alice; tenant_id=acme
```

Session continuity SHOULD be expressed with `X-AAMP-Session-Key` rather than by placing session identifiers in `X-AAMP-Dispatch-Context`. `X-AAMP-TaskId` remains unique to an individual dispatch lifecycle; the session key only hints that multiple dispatches belong to the same higher-level conversation.

### 5.4 Pairing URL and Rule Encoding

A pairing URL uses the `aamp://connect` scheme and carries at least a receiver mailbox and one-time code.

```
aamp://connect?mailbox=agent@meshmail.ai&pair_code=<base64url-code>
aamp://connect?mailbox=agent@meshmail.ai&pair_code=<base64url-code>&dispatch_co
```

`mailbox` identifies the receiver that generated the code. `pair_code` is an opaque base64url token and MUST NOT be interpreted by consumers. `dispatch_context_rules`, when present, is a base64url-encoded UTF-8 JSON object whose keys are dispatch-context keys and whose values are arrays of allowed string values.

---

# 6. Lifecycle Intents

## 6.1 task.dispatch

`task.dispatch` creates a new task or adds clarifying input to an existing task thread. The dispatcher MUST include the required core fields and SHOULD include a clear subject line and narrative body that a human participant can understand.

A receiver MAY treat a follow-up dispatch in the same thread as additional instruction rather than as a distinct child task unless `X-AAMP-ParentTaskId` or local application policy indicates otherwise.

## 6.2 task.ack

`task.ack` confirms that an executor has received and admitted the task into a local processing context. Acknowledgement does not imply completion. An executor SHOULD emit `task.ack` promptly after acceptance, either automatically or explicitly.

## 6.3 task.help_needed

`task.help_needed` indicates that the executor cannot safely continue without additional information, approval, or policy clarification. The human question and blocked reason SHOULD appear in the body. Suggested structured response options MAY appear in `X-AAMP-SuggestedOptions`.

## 6.4 task.result

`task.result` is the authoritative terminal response for the core protocol. A result message MUST carry `X-AAMP-Status`. The current specification defines:

- `completed`: The task was carried out successfully.
- `rejected`: The task could not be accepted or could not be honorably completed as requested.

The human-readable output or rejection explanation SHOULD appear in the body. Structured writeback data MAY be supplied in `X-AAMP-StructuredResult`.

### 6.5 task.cancel

`task.cancel` withdraws a previously dispatched task. After validating that the cancellation belongs to the same task thread, the executor SHOULD suppress any later ordinary completion response if possible and SHOULD stop queued or active work when safe and practical.

### 6.6 pair.request

`pair.request` asks a receiver to authorize the sender for future dispatches. It exists to solve first-contact onboarding: a user can scan or paste a receiver-generated pairing URL instead of manually editing sender-policy configuration or exposing a public callback endpoint.

The sender MUST include `X-AAMP-Pair-Code` and SHOULD use a fresh `X-AAMP-TaskId`. If the pairing URL included dispatch-context rules, the sender MUST forward them unchanged in `X-AAMP-Dispatch-Context-Rules`.

A receiver MAY bypass ordinary sender policy for `pair.request` only. It MUST validate that the pairing code is known, unexpired, and unconsumed before writing any sender policy. A valid request SHOULD add or update the requester in sender policy, attach any supplied dispatch-context rules, and consume the code so it cannot be reused.

### 6.7 pair.respond

`pair.respond` returns the result of a `pair.request` to the requester. It MUST use the same `X-AAMP-TaskId` as the request and MUST include `X-AAMP-Status`. Success uses `completed`. Failure uses `rejected` and SHOULD include `X-AAMP-ErrorMsg` with the reason, such as an expired, unknown, or already consumed code.

---

## 7. Discovery Document

An AAMP service endpoint MUST publish a discovery document at `/.well-known/aamp`. The document MUST be JSON and MUST identify the protocol name, version, advertised intents, and canonical helper API base used by SDK-targeted implementations. SDK-compatible services SHOULD also advertise helper action names and runtime endpoints.

```
{
  "protocol": "aamp",
  "version": "1.1",
  "intents": [
    "task.dispatch",
    "task.cancel",
    "task.ack",
    "task.help_needed",
    "task.result",
    "task.stream.opened",
    "pair.request",
    "pair.respond",
    "card.query",
    "card.response"
  ],
  "capabilities": {
    "stream": {
      "transport": "sse",
      "createAction": "aamp.stream.create",
      "appendAction": "aamp.stream.append",
      "closeAction": "aamp.stream.close",
      "getAction": "aamp.stream.get",
      "subscribeUrlTemplate": "/api/aamp/streams/{streamId}/events"
    }
  },
  "api": {
    "url": "/api/aamp",
    "actions": [
      "aamp.mailbox.check",
      "aamp.mailbox.register",
      "aamp.mailbox.credentials",
      "aamp.mailbox.send",
      "aamp.mailbox.inbox",
      "aamp.mailbox.thread",
      "aamp.push.register",
      "aamp.push.unregister",
      "aamp.stream.create",
      "aamp.stream.append",
      "aamp.stream.close",
      "aamp.stream.get",
      "aamp.directory.upsert",
      "aamp.directory.list",
      "aamp.directory.search"
    ]
  },
  "endpoints": {
    "discovery": "/.well-known/aamp",
    "api": "/api/aamp",
    "jmapSession": "/.well-known/jmap",
    "jmapApi": "/jmap/",
```

```
    "jmapWebSocket": "/jmap/ws"
  }
}
```

Helper actions advertised by `api.url` are deployment-specific unless separately standardized. Their presence does not expand the mandatory conformance surface of this specification.

## 7.1 SDK Compatibility Profile

The current AAMP SDK implementations depend on more than the core wire protocol. They discover `api.url` from the discovery document and then call a concrete helper interface rooted at that URL. Therefore, a service can conform to the core protocol without being automatically compatible with the current SDK family.

A service that claims compatibility with the present SDK ecosystem SHOULD implement the helper actions in this subsection. This profile is informative for the core specification but normative for SDK-targeted interoperability.

### 7.1.1 Discovery Requirements for SDK Compatibility

An SDK-compatible service MUST return `api.url` from `/.well-known/aamp`. It SHOULD also return `api.actions` so independent implementers can discover the concrete helper surface. If stream support is offered, it MUST also return a `capabilities.stream` object with `transport = "sse"` and either the default stream action names or explicit overrides.

### 7.1.2 Mailbox-Scoped Authentication

Authenticated helper actions use HTTP Basic authentication with a mailbox token. In the current profile, that token is equivalent to the base64 encoding of `email:smtpPassword`.

```
Authorization: Basic <mailboxToken>
```

### 7.1.3 Required Helper Actions

| Action | Method | Authentication | Minimum Contract |
|---|---|---|---|
| `aamp.mailbox.check` | GET | None | Accept `email`; return `{ aamp: boolean, domain?: string }`. |

| Action | Method | Authentication | Minimum Contract |
| --- | --- | --- | --- |
| `aamp.mailbox.register` | POST | None | Accept a JSON body with `slug` and optional `description`. Return a registration code suitable for a subsequent credential exchange. |
| `aamp.mailbox.credentials` | GET | None | Accept a query parameter `code`. Return `email`, `mailbox.token`, and `smtp.password`. |
| `aamp.mailbox.send` | POST | Basic mailbox token | Accept `to`, `subject`, `text`, optional `aampHeaders`, and optional base64-encoded attachments. Return a response containing `messageId`. |
| `aamp.mailbox.inbox` | GET | Basic mailbox token | Return mailbox task summaries for the authenticated mailbox. |
| `aamp.mailbox.thread` | GET | Basic mailbox token | Accept `taskId` and optional `includeStreamOpened`. Return `{ taskId, events[] }`. |
| `aamp.stream.create` | POST | Basic mailbox token | Accept `taskId` and `peerEmail`. Return stream state including `streamId`. |
| `aamp.stream.append` | POST | Basic mailbox token | Accept `streamId`, `type`, and `payload`. Return the appended event. |
| `aamp.stream.close` | POST | Basic mailbox token | Accept `streamId` and optional payload. Return the final stream state. |
| `aamp.stream.get` | GET | Basic mailbox token | Accept `streamId` or `taskId`. Return the current stream state or 404. |

| Action | Method | Authentication | Minimum Contract |
| --- | --- | --- | --- |
| `aamp.directory.upsert` | POST | Basic mailbox token | Accept `summary` and/or `cardText`; return `{ ok, profile }`. |
| `aamp.directory.list` | GET | Basic mailbox token | Accept optional `scope`, `includeSelf`, and `limit`; return `{ scope, agents }`. |
| `aamp.directory.search` | GET | Basic mailbox token | Accept `q` plus optional `scope`, `includeSelf`, and `limit`; return `{ scope, query, agents }`. |
| `aamp.push.register` / `aamp.push.unregister` | POST | Basic mailbox token | If advertised, accept `deviceToken` and optional `environment`; return `{ ok: true }`. |

#### 7.1.4 Canonical Helper Invocation

SDK-compatible helper actions are invoked against the discovered `api.url` by placing the action name in the `action` query parameter. The action name is not encoded as a REST path segment and is not required in the JSON body. POST bodies contain only the action arguments. GET arguments are query parameters alongside `action`. Relative `api.url` values are resolved against the discovery origin.

```
POST /api/aamp?action=aamp.mailbox.register
Content-Type: application/json

{ "slug": "demo-agent", "description": "Demo AAMP agent" }

GET /api/aamp?action=aamp.mailbox.credentials&code=<registrationCode>

POST /api/aamp?action=aamp.mailbox.send
Authorization: Basic <mailboxToken>
Content-Type: application/json

{
  "to": "worker@example.com",
  "subject": "Review task",
  "text": "Please review this change.",
  "aampHeaders": {
    "X-AAMP-Version": "1.1",
```

```
    "X-AAMP-Intent": "task.dispatch",
    "X-AAMP-TaskId": "task_123"
  },
  "attachments": [
    { "filename": "context.txt", "contentType": "text/plain", "content": "<base
  ]
}
```

#### 7.1.5 SDK Response Contracts

| Surface | Response Contract |
|---|---|
| `aamp.mailbox.register` | Return HTTP 201 or 200 with `{ id, email, description, registrationCode, expiresInSeconds, credentialsAction: "aamp.mailbox.credentials" }`. `registrationCode` is the required field name consumed by SDK registration. |
| `aamp.mailbox.credentials` | Return `{ email, mailbox: { token }, smtp: { password } }`. The mailbox token is sent later as `Authorization: Basic <token>`. |
| `aamp.mailbox.send` | Request attachments use `{ filename, contentType, content }` where `content` is base64. Return `{ ok: true, from, to, messageId }`. |
| `aamp.mailbox.thread` | Return `{ taskId, events }`. Each event should include `intent`, `from`, `to`, `createdAt`, and may include `status`, `title`, `bodyText`, `output`, `errorMsg`, `question`, `blockedReason`, `attachments`, and `messageId`. |
| `aamp.stream.create` | Return `TaskStreamState`: `{ streamId, taskId, ownerEmail, peerEmail, status, createdAt, openedAt?, closedAt?, latestEvent? }`, where `status` is `created`, `opened`, or `closed`. |
| `aamp.stream.append` | Return a `StreamEvent`: `{ id, streamId, taskId, seq, timestamp, type, payload }`. `seq` is monotonically increasing within the stream. |
| `aamp.stream.close` | Append a final `done` event when practical and return the final `TaskStreamState` with `status: "closed"`. |
| `aamp.stream.get` | Return the current `TaskStreamState` by `streamId` or latest stream for `taskId`, or 404 when absent. |

| Surface | Response Contract |
| --- | --- |
| SSE subscribe URL | Resolve `capabilities.stream.subscribeUrlTemplate`, replace `{streamId}`, require Basic auth, support `Last-Event-ID` or `lastEventId`, and emit SSE frames with `id: <event.id>`, `event: <event.type>`, and `data:` containing the full `StreamEvent` JSON. |
| Directory actions | `upsert` returns `{ ok: true, profile: { email, summary, cardText } }`; `list` returns `{ scope, agents: [{ email, summary }] }`; `search` returns `{ scope, query, agents: [{ email, summary, score }] }`. |

#### 7.1.6 Extended SDK Surface

The current SDK also exposes directory methods. Full feature-level compatibility SHOULD additionally implement `aamp.directory.upsert`, `aamp.directory.list`, and `aamp.directory.search`. Management-facing deployments may also implement `aamp.mailbox.check` and `aamp.mailbox.inbox`.

### 7.2 Full Runtime Compatibility Profile

SDK helper compatibility alone is insufficient to reproduce the behavior of a deployed `AampClient`. A service that claims full runtime compatibility MUST additionally expose the mailbox retrieval, push, blob, and submission surfaces used by `connect()`, attachment download, and Sent-mail projection.

#### 7.2.1 Required Service Surface

A full runtime-compatible service SHOULD advertise the following endpoints from `/.well-known/aamp` and MUST serve them at stable URLs on the discovered origin:

- `/.well-known/jmap` for authenticated JMAP session discovery.
- `/jmap/` for JMAP method calls. Servers SHOULD also accept `/jmap` without the trailing slash for compatibility, but SDK clients call the slash form.
- `/jmap/ws` for JMAP-over-WebSocket push.
- `/jmap/download/{accountId}/{blobId}/{name}` when the JMAP session does not provide `downloadUrl`.

- SMTP submission, or a standards-equivalent authenticated submission path, for the provisioned mailbox identity.

#### 7.2.2 JMAP Session and Method Requirements

The service MUST accept `Authorization: Basic <mailboxToken>` on `GET /.well-known/jmap` and return a valid JMAP session object containing at least `accounts`, `primaryAccounts`, `username`, `apiUrl`, and `state`. `apiUrl` SHOULD resolve to `/jmap/`; implementations MAY additionally accept `/jmap` as an alias. A usable `downloadUrl` is RECOMMENDED.

The service MUST support the following JMAP mail methods on `POST /jmap/`:

| Method | Minimum Requirement |
|---|---|
| `Email/get` | Support empty `ids` to obtain state. For fetched messages, include `id`, `blobId`, `threadId`, `mailboxIds`, `keywords`, `size`, `receivedAt`, `messageId`, `from`, `to`, `subject`, `headers`, `textBody`, `bodyValues`, and `attachments`. `messageId` MUST be an array of Message-ID strings. `headers` MUST be an array of `{ name, value }` entries containing all `X-AAMP-*` headers. The first `textBody[].partId` MUST have a matching `bodyValues[partId].value`, because SDK receivers read body text through that relation. |
| `Email/changes` | Support incremental sync by `sinceState` or return the standard error indicating that state-based changes cannot be calculated. |
| `Email/query` | Support recent-message queries sorted by `receivedAt` descending. |
| `Mailbox/get` | Expose mailbox roles so the client can locate a Sent mailbox. |
| `Email/set` | Allow create operations for Sent-copy projection, including arbitrary `header:<name>:asText` properties. |

#### 7.2.3 WebSocket Push and Polling Fallback

A push-complete service SHOULD implement RFC 8887 at `/jmap/ws`, accept the `jmap` subprotocol, and process a `WebSocketPushEnable` request for the `Email` data type. It SHOULD emit `StateChange` events when mailbox state changes.

A service without WebSocket push can still satisfy runtime compatibility if the JMAP HTTP methods above are implemented correctly, because the current SDK falls back to `Email/query` and `Email/changes` polling. Such a service is runtime compatible but not push-complete.

#### 7.2.4 Blob and Attachment Requirements

Attachment descriptors returned from `Email/get` MUST include `blobId`, `type`, `name`, and `size`. The service SHOULD expose a session `downloadUrl`; otherwise it MUST serve the fallback download path on the discovered origin and protect it with the same mailbox authentication model.

#### 7.2.5 Submission and Sent Projection

The credentials returned by `aamp.mailbox.credentials` MUST be sufficient for authenticated message submission. The current profile binds `mailbox.token` and `smtp.password` to the same mailbox identity. Services SHOULD support both standards-aligned SMTP submission and the same-origin helper send path `aamp.mailbox.send`. Services that claim full runtime compatibility SHOULD also make Sent copies visible through the required `Mailbox/get` and `Email/set` support described above.

---

# 8. Processing Model

## 8.1 Sender Requirements

- A sender MUST emit the required core fields for every AAMP message.
- A sender SHOULD supply a globally unique `Message-ID`.
- A sender SHOULD preserve thread references when replying within a task thread.
- A sender SHOULD place human-facing narrative content in the body rather than in extension fields.

## 8.2 Receiver Requirements

- A receiver MUST parse header names case-insensitively.
- A receiver MUST ignore unknown extensions it does not understand.
- A receiver SHOULD apply de-duplication based on message identity and local delivery state.
- A receiver SHOULD honor `X-AAMP-Expires-At` when rebuilding pending work after downtime.
- A receiver SHOULD update local task state when valid `task.cancel` is received.

- A receiver MUST treat `pair.request` as a constrained authorization flow: validate the one-time code before policy changes, consume valid codes after use, and reject invalid requests with `pair.respond`.

### 8.3 Local State Projection

Implementations may project richer internal state machines than those represented by the wire protocol. Common local states include pending, running, help_needed, cancelled, expired, or failed. Such states are useful operationally but are not, by themselves, additional wire-level intents in the core specification.

---

## 9. Streaming Observation Extension

### 9.1 Scope

The streaming observation extension allows an executor to expose incremental progress for a task while preserving mail as the authoritative control plane.

### 9.2 task.stream.opened

A stream-capable executor MAY send `task.stream.opened` after creating a stream resource for a task. The message MUST include `X-AAMP-Stream-Id`.

### 9.3 Discovery and Subscription

The discovery document SHOULD advertise stream capability metadata, including transport type and a subscription URL template. Stream implementations SHOULD support replay after reconnect through event identifiers or equivalent cursor semantics.

### 9.4 Recommended Event Types

Implementations commonly expose `text.delta`, `progress`, `status`, `artifact`, `error`, and `done`. This document does not mandate a complete payload schema for each event type, but future standards work may define one.

---

## 10. Security Considerations

AAMP relies on the underlying mail and web infrastructure for transport security, sender authentication, and credential handling. Implementations SHOULD use TLS for mail submission and HTTPS for helper or stream endpoints.

Dispatch context is not identity. A receiver MUST NOT rely solely on `X-AAMP-Dispatch-Context` for authenticity or authorization.

Pairing codes are authorization tokens. Implementations SHOULD generate them with cryptographically secure randomness, SHOULD use short expirations, MUST consume them after successful use, and SHOULD avoid logging complete pairing URLs in shared logs. A `pair.request` bypasses ordinary sender policy only for code validation; it MUST NOT become a general unauthenticated dispatch path.

In the current reference deployment, external sender trust is grounded in successful DKIM verification at the mail transport layer. This specification does not mandate DKIM specifically, but it does require some equivalent transport-authenticated trust basis.

---

# 11. Privacy Considerations

Task bodies, attachments, and structured result payloads may contain sensitive information. Implementations SHOULD avoid placing personal or business-sensitive data in headers unless required for routing, and SHOULD apply mailbox retention, access control, encryption, and audit policies appropriate to their environment.

---

# 12. Registry and Standardization Considerations

The following items are likely candidates for formal registration in a future standards process.

- The `/.well-known/aamp` discovery resource.
- The AAMP header field set.
- The core lifecycle intent registry.
- An extension registry for optional capabilities and stream event types.

---

# 13. Examples

## 13.1 Dispatch Example

```
Subject: [AAMP Task] Summarize release notes
X-AAMP-Version: 1.1
```

```
X-AAMP-Intent: task.dispatch
X-AAMP-TaskId: 9f0f4a9a-2d3a-4f68-a430-2f4548cda52f
X-AAMP-Priority: normal
X-AAMP-Dispatch-Context: project_key=release_42; user_key=alice

Please summarize the attached release notes and return three operator-facing bu
```

### 13.2 Result Example

```
Subject: [AAMP Result] Task 9f0f4a9a-2d3a-4f68-a430-2f4548cda52f - completed
X-AAMP-Version: 1.1
X-AAMP-Intent: task.result
X-AAMP-TaskId: 9f0f4a9a-2d3a-4f68-a430-2f4548cda52f
X-AAMP-Status: completed

Output:
1. Release improves mailbox sync resilience.
2. Streaming status handling is now clearer.
3. Operator setup is simplified for new nodes.
```

### 13.3 Pairing Example

```
aamp://connect?mailbox=openclaw-agent-7d137b5e@meshmail.ai&pair_code=7r8g5A2k
```

The consumer sends:

```
Subject: [AAMP Pair] Request sender authorization
X-AAMP-Version: 1.1
X-AAMP-Intent: pair.request
X-AAMP-TaskId: 4c9d0c91-4a2f-4b6b-9df7-75cfd4bb7fd4
X-AAMP-Pair-Code: 7r8g5A2k

Please authorize this mailbox as a sender.
```

If accepted, the receiver replies:

```
Subject: [AAMP Pair] completed
X-AAMP-Version: 1.1
X-AAMP-Intent: pair.respond
X-AAMP-TaskId: 4c9d0c91-4a2f-4b6b-9df7-75cfd4bb7fd4
X-AAMP-Status: completed

Sender policy updated.
```

# 14. References

## 14.1 Normative References

- RFC 2119, Key words for use in RFCs to Indicate Requirement Levels.
- RFC 8174, Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words.
- RFC 5322, Internet Message Format.
- RFC 8620, The JSON Meta Application Protocol (JMAP).
- RFC 8621, JMAP for Mail.

## 14.2 Informative References

- RFC 8887, JMAP over WebSocket.
- RFC 6376, DomainKeys Identified Mail (DKIM) Signatures.

End of Specification.