Agent2Agent (A2A) Protocol Specification

Version: `0.2.2`

1. Introduction

The Agent2Agent (A2A) Protocol is an open standard designed to facilitate communication and interoperability between independent, potentially opaque AI agent systems. In an ecosystem where agents might be built using different frameworks, languages, or by different vendors, A2A provides a common language and interaction model.

This document provides the detailed technical specification for the A2A protocol. Its primary goal is to enable agents to:

- Discover each other's capabilities.
- Negotiate interaction modalities (text, files, structured data).
- Manage collaborative tasks.
- Securely exchange information to achieve user goals **without needing access to each other's internal state, memory, or tools.**

1.1. Key Goals of A2A

- **Interoperability:** Bridge the communication gap between disparate agentic systems.
- **Collaboration:** Enable agents to delegate tasks, exchange context, and work together on complex user requests.
- **Discovery:** Allow agents to dynamically find and understand the capabilities of other agents.
- **Flexibility:** Support various interaction modes including synchronous request/response, streaming for real-time updates, and asynchronous push notifications for long-running tasks.

- **Security:** Facilitate secure communication patterns suitable for enterprise environments, relying on standard web security practices.
- **Asynchronicity:** Natively support long-running tasks and interactions that may involve human-in-the-loop scenarios.

1.2. Guiding Principles

- **Simple:** Reuse existing, well-understood standards (HTTP, JSON-RPC 2.0, Server-Sent Events).
- **Enterprise Ready:** Address authentication, authorization, security, privacy, tracing, and monitoring by aligning with established enterprise practices.
- **Async First:** Designed for (potentially very) long-running tasks and human-in-the-loop interactions.
- **Modality Agnostic:** Support exchange of diverse content types including text, audio/video (via file references), structured data/forms, and potentially embedded UI components (e.g., iframes referenced in parts).
- **Opaque Execution:** Agents collaborate based on declared capabilities and exchanged information, without needing to share their internal thoughts, plans, or tool implementations.

For a broader understanding of A2A's purpose and benefits, see What is A2A?.

2. Core Concepts Summary

A2A revolves around several key concepts. For detailed explanations, please refer to the Key Concepts guide.

- **A2A Client:** An application or agent that initiates requests to an A2A Server on behalf of a user or another system.
- **A2A Server (Remote Agent):** An agent or agentic system that exposes an A2A-compliant HTTP endpoint, processing tasks and providing responses.
- **Agent Card:** A JSON metadata document published by an A2A Server, describing its identity, capabilities, skills, service endpoint, and

authentication requirements.

- **Message:** A communication turn between a client and a remote agent, having a `role` ("user" or "agent") and containing one or more `Parts`.
- **Task:** The fundamental unit of work managed by A2A, identified by a unique ID. Tasks are stateful and progress through a defined lifecycle.
- **Part:** The smallest unit of content within a Message or Artifact (e.g., `TextPart`, `FilePart`, `DataPart`).
- **Artifact:** An output (e.g., a document, image, structured data) generated by the agent as a result of a task, composed of `Parts`.
- **Streaming (SSE):** Real-time, incremental updates for tasks (status changes, artifact chunks) delivered via Server-Sent Events.
- **Push Notifications:** Asynchronous task updates delivered via server-initiated HTTP POST requests to a client-provided webhook URL, for long-running or disconnected scenarios.
- **Context:** An optional, server-generated identifier to logically group related tasks.
- **Extension:** A mechanism for agents to provide additional functionality or data beyond the core A2A specification.

3. Transport and Format

3.1. Transport Protocol

- A2A communication **MUST** occur over **HTTP(S)**.
- The A2A Server exposes its service at a URL defined in its `AgentCard`.

3.2. Data Format

A2A uses **JSON-RPC 2.0** as the payload format for all requests and responses (excluding the SSE stream wrapper).

- Client requests and server responses **MUST** adhere to the JSON-RPC 2.0 specification.
- The `Content-Type` header for HTTP requests and responses containing JSON-RPC payloads **MUST** be `application/json`.

3.3. Streaming Transport (Server-Sent Events)

When streaming is used for methods like `message/stream` or `tasks/resubscribe`:

- The server responds with an HTTP `200 OK` status and a `Content-Type` header of `text/event-stream`.
- The body of this HTTP response contains a stream of **Server-Sent Events (SSE)** as defined by the W3C.
- Each SSE `data` field contains a complete JSON-RPC 2.0 Response object (specifically, a `SendStreamingMessageResponse`).

4. Authentication and Authorization

A2A treats agents as standard enterprise applications, relying on established web security practices. Identity information is **not** transmitted within A2A JSON-RPC payloads; it is handled at the HTTP transport layer.

For a comprehensive guide on enterprise security aspects, see Enterprise-Ready Features.

4.1. Transport Security

As stated in section 3.1, production deployments **MUST** use HTTPS. Implementations **SHOULD** use modern TLS configurations (TLS 1.2+ recommended) with strong cipher suites.

4.2. Server Identity Verification

A2A Clients **SHOULD** verify the A2A Server's identity by validating its TLS certificate against trusted certificate authorities (CAs) during the TLS handshake.

4.3. Client/User Identity & Authentication Process

1. **Discovery of Requirements:** The client discovers the server's required authentication schemes via the `authentication` field in the `AgentCard`. Scheme names often align with OpenAPI Authentication methods (e.g.,

"Bearer" for OAuth 2.0 tokens, "Basic" for Basic Auth, "ApiKey" for API keys).

2. **Credential Acquisition (Out-of-Band):** The client obtains the necessary credentials (e.g., API keys, OAuth tokens, JWTs) through an **out-of-band process** specific to the required authentication scheme and the identity provider. This process is outside the scope of the A2A protocol itself.
3. **Credential Transmission:** The client includes these credentials in the appropriate HTTP headers (e.g., `Authorization: Bearer <token>`, `X-API-Key: <value>`) of every A2A request sent to the server.

4.4. Server Responsibilities for Authentication

The A2A Server:

- **MUST** authenticate every incoming request based on the provided HTTP credentials and its declared authentication requirements from its Agent Card.
- **SHOULD** use standard HTTP status codes like `401 Unauthorized` or `403 Forbidden` for authentication challenges or rejections.
- **SHOULD** include relevant HTTP headers (e.g., `WWW-Authenticate`) with `401 Unauthorized` responses to indicate the required authentication scheme(s), guiding the client.

4.5. In-Task Authentication (Secondary Credentials)

If an agent, during the execution of a task, requires *additional* credentials for a *different* system or resource (e.g., to access a specific tool on behalf of the user that requires its own auth):

1. It **SHOULD** transition the A2A task to the `auth-required` state (see `TaskState`).
2. The accompanying `TaskStatus.message` (often a `DataPart`) **SHOULD** provide details about the required secondary authentication, potentially using an `PushNotificationAuthenticationInfo`-like structure to describe the need.
3. The A2A Client then obtains these new credentials out-of-band and provides them in a subsequent `message/send` or `message/stream` request. How these credentials are used (e.g., passed as data within the

A2A message if the agent is proxying, or used by the client to interact directly with the secondary system) depends on the specific scenario.

4.6. Authorization

Once a client is authenticated, the A2A Server is responsible for authorizing the request based on the authenticated client/user identity and its own policies. Authorization logic is implementation-specific and MAY be enforced based on:

- The specific skills requested (e.g., as identified by `AgentSkill.id` from the Agent Card).
- The actions attempted within the task.
- Data access policies relevant to the resources the agent manages.
- OAuth scopes associated with the presented token, if applicable.

Servers should implement the principle of least privilege.

5. Agent Discovery: The Agent Card

5.1. Purpose

A2A Servers **MUST** make an Agent Card available. The Agent Card is a JSON document that describes the server's identity, capabilities, skills, service endpoint URL, and how clients should authenticate and interact with it. Clients use this information for discovering suitable agents and for configuring their interactions.

For more on discovery strategies, see the Agent Discovery guide.

5.2. Discovery Mechanisms

Clients can find Agent Cards through various methods, including but not limited to:

- **Well-Known URI:** Accessing a predefined path on the agent's domain (see Section 5.3).
- **Registries/Catalogs:** Querying curated catalogs or registries of agents (which might be enterprise-specific, public, or domain-specific).

- **Direct Configuration:** Clients may be pre-configured with the Agent Card URL or the card content itself.

5.3. Recommended Location

If using the well-known URI strategy, the recommended location for an agent's Agent Card is: `https://{server_domain}/.well-known/agent.json`
This follows the principles of RFC 8615 for well-known URIs.

5.4. Security of Agent Cards

Agent Cards themselves might contain information that is considered sensitive.

- If an Agent Card contains sensitive information, the endpoint serving the card **MUST** be protected by appropriate access controls (e.g., mTLS, network restrictions, authentication required to fetch the card).
- It is generally **NOT RECOMMENDED** to include plaintext secrets (like static API keys) directly in an Agent Card. Prefer authentication schemes where clients obtain dynamic credentials out-of-band.

5.5. `AgentCard` Object Structure

```
/**
 * An AgentCard conveys key information:
 * - Overall details (version, name, description, uses)
 * - Skills: A set of capabilities the agent can perform
 * - Default modalities/content types supported by the agent.
 * - Authentication requirements
 */
export interface AgentCard {
  /**
   * Human readable name of the agent.
   * @example "Recipe Agent"
   */
  name: string;
  /**
   * A human-readable description of the agent. Used to assist
users and
   * other agents in understanding what the agent can do.
   * @example "Agent that helps users with recipes and
cooking."
   */
  description: string;
  /** A URL to the address the agent is hosted at. */
```

```
  url: string;
  /** A URL to an icon for the agent. */
  iconUrl?: string;
  /** The service provider of the agent */
  provider?: AgentProvider;
  /**
   * The version of the agent - format is up to the provider.
   * @example "1.0.0"
   */
  version: string;
  /** A URL to documentation for the agent. */
  documentationUrl?: string;
  /** Optional capabilities supported by the agent. */
  capabilities: AgentCapabilities;
  /** Security scheme details used for authenticating with this
agent. */
  securitySchemes?: { [scheme: string]: SecurityScheme };
  /** Security requirements for contacting the agent. */
  security?: { [scheme: string]: string[] }[];
  /**
   * The set of interaction modes that the agent supports
across all skills. This can be overridden per-skill.
   * Supported media types for input.
   */
  defaultInputModes: string[];
  /** Supported media types for output. */
  defaultOutputModes: string[];
  /** Skills are a unit of capability that an agent can
perform. */
  skills: AgentSkill[];
  /**
   * true if the agent supports providing an extended agent
card when the user is authenticated.
   * Defaults to false if not specified.
   */
  supportsAuthenticatedExtendedCard?: boolean;
}
```

Field Name	Type	Required	Description
`name`	`string`	Yes	Human-readable name of the agent.
`descriptio n`	`string`	Yes	Human-readable description. CommonMark MAY be used.

Field Name	Type	Required	Description
`url`	`string`	Yes	Base URL for the agent's A2A service. Must be absolute. HTTPS for production.
`provider`	`AgentProvider`	No	Information about the agent's provider.
`iconUrl`	`string`	No	A URL to an icon for the agent.
`version`	`string`	Yes	Agent or A2A implementation version string.
`documentationUrl`	`string`	No	URL to human-readable documentation for the agent.
`capabilities`	`AgentCapabilities`	Yes	Specifies optional A2A protocol features supported (e.g., streaming, push notifications).
`securitySchemes`	{ [scheme: string]: SecurityScheme }	No	Security scheme details used for authenticating with this agent. undefined implies no A2A-advertised auth (not

Field Name	Type	Required	Description
			recommended for production).
`security`	`{ [scheme: string]: string[]; }[]`	No	Security requirements for contacting the agent.
`defaultInputModes`	`string[]`	Yes	Input Media Types accepted by the agent.
`defaultOutputModes`	`string[]`	Yes	Output Media Types produced by the agent.
`skills`	`AgentSkill[]`	Yes	Array of skills. Must have at least one if the agent performs actions.
`supportsAuthenticatedExtendedCard`	`boolean`	No	Indicates support for retrieving a more detailed Agent Card via an authenticated endpoint.

5.5.1. `AgentProvider` Object

Information about the organization or entity providing the agent.

```
/**
 * Represents the service provider of an agent.
 */
export interface AgentProvider {
  /** Agent provider's organization name. */
  organization: string;
  /** Agent provider's URL. */
  url: string;
```

```
}
```

Field Name	Type	Required	Description
`organization`	`string`	Yes	Name of the organization/entity.
`url`	`string`	Yes	URL for the provider's website/contact.

5.5.2. `AgentCapabilities` Object

Specifies optional A2A protocol features supported by the agent.

```
/**
 * Defines optional capabilities supported by an agent.
 */
export interface AgentCapabilities {
  /** true if the agent supports SSE. */
  streaming?: boolean;
  /** true if the agent can notify updates to client. */
  pushNotifications?: boolean;
  /** true if the agent exposes status change history for
tasks. */
  stateTransitionHistory?: boolean;
  /** extensions supported by this agent. */
  extensions?: AgentExtension[];
}
```

Field Name	Type	Required	Default	Description
`streaming`	`boolean`	No	`false`	Indicates support for SSE streaming methods (`message/stream`, `tasks/resubscribe`).

Field Name	Type	Required	Default	Description
`pushNotifications`	`boolean`	No	`false`	Indicates support for push notification methods (`tasks/pushNotificationConfig/*`).
`stateTransitionHistory`	`boolean`	No	`false`	Placeholder for future feature: exposing detailed task status change history.
`extensions`	`AgentExtension[]`	No	`[]`	A list of extensions supported by this agent.

5.5.2.1. `AgentExtension` Object

Specifies an extension to the A2A protocol supported by the agent.

```ts
/**
 * A declaration of an extension supported by an Agent.
 */
export interface AgentExtension {
  /** The URI of the extension. */
  uri: string;
  /** A description of how this agent uses this extension. */
  description?: string;
```

```
  /** Whether the client must follow specific requirements of
the extension. */
  required?: boolean;
  /** Optional configuration for the extension. */
  params?: { [key: string]: any };
}
```

Field Name	Type	Required	Description
`uri`	`string`	Yes	The URI for the supported extension.
`required`	`boolean`	No	Whether the agent requires clients to follow some protocol logic specific to the extension. Clients should expect failures when attempting to interact with a server that requires an extension the client does not support.
`descriptio n`	`string`	No	A description of how the extension is used by the agent.
`params`	`object`	No	Configuration parameters specific to the extension

5.5.3. `SecurityScheme` Object

Describes the authentication requirements for accessing the agent's `url` endpoint. Refer Sample Agent Card for an example.

```
/**
 * Mirrors the OpenAPI Security Scheme Object
 * (https://swagger.io/specification/#security-scheme-object)
 */
export type SecurityScheme =
```

```
  | APIKeySecurityScheme
  | HTTPAuthSecurityScheme
  | OAuth2SecurityScheme
  | OpenIdConnectSecurityScheme;
```

5.5.4. `AgentSkill` Object

Describes a specific capability, function, or area of expertise the agent can perform or address.

```
/**
 * Represents a unit of capability that an agent can perform.
 */
export interface AgentSkill {
  /** Unique identifier for the agent's skill. */
  id: string;
  /** Human readable name of the skill. */
  name: string;
  /**
   * Description of the skill - will be used by the client or a
human
   * as a hint to understand what the skill does.
   */
  description: string;
  /**
   * Set of tagwords describing classes of capabilities for
this specific skill.
   * @example ["cooking", "customer support", "billing"]
   */
  tags: string[];
  /**
   * The set of example scenarios that the skill can perform.
   * Will be used by the client as a hint to understand how the
skill can be used.
   * @example ["I need a recipe for bread"]
   */
  examples?: string[]; // example prompts for tasks
  /**
   * The set of interaction modes that the skill supports
   * (if different than the default).
   * Supported media types for input.
   */
  inputModes?: string[];
  /** Supported media types for output. */
  outputModes?: string[];
}
```

Field Name	Type	Required	Description
`id`	`string`	Yes	Unique skill identifier within this agent.
`name`	`string`	Yes	Human-readable skill name.
`description`	`string`	Yes	Detailed skill description. CommonMark MAY be used.
`tags`	`string[]`	Yes	Keywords/categories for discoverability.
`examples`	`string[]`	No	Example prompts or use cases demonstrating skill usage.
`inputModes`	`string[]`	No	Overrides `defaultInputModes` for this specific skill. Accepted Media Types.
`outputModes`	`string[]`	No	Overrides `defaultOutputModes` for this specific skill. Produced Media Types.

5.6. Sample Agent Card

```
{
  "name": "GeoSpatial Route Planner Agent",
  "description": "Provides advanced route planning, traffic
analysis, and custom map generation services. This agent can
calculate optimal routes, estimate travel times considering
real-time traffic, and create personalized maps with points of
interest.",
```

```
  "url": "https://georoute-agent.example.com/a2a/v1",
  "provider": {
    "organization": "Example Geo Services Inc.",
    "url": "https://www.examplegeoservices.com"
  },
  "iconUrl": "https://georoute-agent.example.com/icon.png",
  "version": "1.2.0",
  "documentationUrl":
"https://docs.examplegeoservices.com/georoute-agent/api",
  "capabilities": {
    "streaming": true,
    "pushNotifications": true,
    "stateTransitionHistory": false
  },
  "securitySchemes": {
    "google": {
      "type": "openIdConnect",
      "openIdConnectUrl": "https://accounts.google.com/.well-
known/openid-configuration"
    }
  },
  "security": [{ "google": ["openid", "profile", "email"] }],
  "defaultInputModes": ["application/json", "text/plain"],
  "defaultOutputModes": ["application/json", "image/png"],
  "skills": [
    {
      "id": "route-optimizer-traffic",
      "name": "Traffic-Aware Route Optimizer",
      "description": "Calculates the optimal driving route
between two or more locations, taking into account real-time
traffic conditions, road closures, and user preferences (e.g.,
avoid tolls, prefer highways).",
      "tags": ["maps", "routing", "navigation", "directions",
"traffic"],
      "examples": [
        "Plan a route from '1600 Amphitheatre Parkway, Mountain
View, CA' to 'San Francisco International Airport' avoiding
tolls.",
        "{\"origin\": {\"lat\": 37.422, \"lng\": -122.084},
\"destination\": {\"lat\": 37.7749, \"lng\": -122.4194},
\"preferences\": [\"avoid_ferries\"]}"
      ],
      "inputModes": ["application/json", "text/plain"],
      "outputModes": [
        "application/json",
        "application/vnd.geo+json",
        "text/html"
      ]
    },
    {
      "id": "custom-map-generator",
      "name": "Personalized Map Generator",
```

```
      "description": "Creates custom map images or interactive
map views based on user-defined points of interest, routes, and
style preferences. Can overlay data layers.",
      "tags": ["maps", "customization", "visualization",
"cartography"],
      "examples": [
        "Generate a map of my upcoming road trip with all
planned stops highlighted.",
        "Show me a map visualizing all coffee shops within a 1-
mile radius of my current location."
      ],
      "inputModes": ["application/json"],
      "outputModes": [
        "image/png",
        "image/jpeg",
        "application/json",
        "text/html"
      ]
    }
  ],
  "supportsAuthenticatedExtendedCard": true
}
```

6. Protocol Data Objects

These objects define the structure of data exchanged within the JSON-RPC methods of the A2A protocol.

6.1. `Task` Object

Represents the stateful unit of work being processed by the A2A Server for an A2A Client. A task encapsulates the entire interaction related to a specific goal or request.

```
export interface Task {
  /** Unique identifier for the task */
  id: string;
  /** Server-generated id for contextual alignment across
interactions */
  contextId: string;
  /** Current status of the task */
  status: TaskStatus;
  history?: Message[];
  /** Collection of artifacts created by the agent. */
  artifacts?: Artifact[];
  /** Extension metadata. */
  metadata?: {
    [key: string]: any;
```

```
  };
  /** Event type */
  kind: "task";
}
```

Field Name	Type	Required	Description
`id`	`string`	Yes	Server generated unique task identifier (e.g., UUID)
`contextId`	`string`	Yes	Server generated ID for contextual alignment across interactions
`status`	`TaskStatus`	Yes	Current status of the task (state, message, timestamp).
`artifacts`	`Artifact[]`	No	Array of outputs generated by the agent for this task.
`history`	`Message[]`	No	Optional array of recent messages exchanged, if requested by `historyLength`.
`metadata`	`Record<string, any>`	No	Arbitrary key-value metadata associated with the task.

6.2. `TaskStatus` Object

Represents the current state and associated context (e.g., a message from the agent) of a `Task`.

```ts
/** TaskState and accompanying message. */
export interface TaskStatus {
  state: TaskState;
  /** Additional status updates for client */
  message?: Message;
  /**
   * ISO 8601 datetime string when the status was recorded.
   * @example "2023-10-27T10:00:00Z"
   * */
  timestamp?: string;
}
```

Field Name	Type	Required	Description
`state`	`TaskState`	Yes	Current lifecycle state of the task.
`message`	`Message`	No	Optional message providing context for the current status.
`timestamp`	`string` (ISO 8601)	No	Timestamp (UTC recommended) when this status was recorded.

6.3. `TaskState` Enum

Defines the possible lifecycle states of a `Task`.

```ts
/** Represents the possible states of a Task. */
export enum TaskState {
  Submitted = "submitted",
  Working = "working",
  InputRequired = "input-required",
  Completed = "completed",
  Canceled = "canceled",
  Failed = "failed",
  Rejected = "rejected",
  AuthRequired = "auth-required",
  Unknown = "unknown",
}
```

Value	Description	Terminal?
`submitted`	Task received by the server and acknowledged, but processing has not yet actively started.	No
`working`	Task is actively being processed by the agent. Client may expect further updates or a terminal state.	No
`input-required`	Agent requires additional input from the client/user to proceed. The task is effectively paused.	No (Pause)
`completed`	Task finished successfully. Results are typically available in `Task.artifacts` or `TaskStatus.message`.	Yes
`canceled`	Task was canceled (e.g., by a `tasks/cancel` request or server-side policy).	Yes
`failed`	Task terminated due to an error during processing. `TaskStatus.message` may contain error details.	Yes
`rejected`	Task terminated due to rejection by remote agent. `TaskStatus.message` may contain error details.	Yes
`auth-required`	Agent requires additional authentication from the client/user to proceed. The task is effectively paused.	No (Pause)
`unknown`	The state of the task cannot be determined (e.g., task ID is invalid, unknown, or has expired).	Yes

6.4. `Message` Object

Represents a single communication turn or a piece of contextual information between a client and an agent. Messages are used for instructions, prompts, replies, and status updates.

```
/** Represents a single message exchanged between user and
agent. */
export interface Message {
  /** Message sender's role */
  role: "user" | "agent";
  /** Message content */
  parts: Part[];
  /** Extension metadata. */
  metadata?: {
    [key: string]: any;
  };
  /** The URIs of extensions that are present or contributed to
this Message. */
  extensions?: string[];
  /** List of tasks referenced as context by this message.*/
  referenceTaskIds?: string[];
  /** Identifier created by the message creator*/
  messageId: string;
  /** Identifier of task the message is related to */
  taskId?: string;
  /** The context the message is associated with */
  contextId?: string;
  /** Event type */
  kind: "message";
}
```

Field Name	Type	Required	Description
`role`	`"user"` | `"agent"`	Yes	Indicates the sender: `"user"` (from A2A Client) or `"agent"` (from A2A Server).
`parts`	`Part[]`	Yes	Array of content parts. Must contain at least one part.

Field Name	Type	Required	Description
`metadata`	`Record<string, any>`	No	Arbitrary key-value metadata associated with this message.
`extensions`	`string[]`	No	A list of extension URIs that contributed to this message.
`referenceTaskIds`	`string[]`	No	List of tasks referenced as contextual hint by this message.
`messageId`	`string`	Yes	Message identifier generated by the message sender
`taskId`	`string`	No	Task identifier the current message is related to
`contextId`	`string`	No	Context identifier the message is associated with
`kind`	`"message"`	Yes	Type discriminator, literal value

6.5. `Part` Union Type

Represents a distinct piece of content within a `Message` or `Artifact`. A `Part` is a union type representing exportable content as either `TextPart`, `FilePart`, or `DataPart`. All `Part` types also include an optional `metadata` field (`Record<string, any>`) for part-specific metadata.

```
/** Represents a part of a message, which can be text, a file,
or structured data. */
```

```
export type Part = TextPart | FilePart | DataPart;
```

It **MUST** be one of the following:

6.5.1. `TextPart` Object

For conveying plain textual content.

```
/** Represents a text segment within parts.*/
export interface TextPart extends PartBase {
  /** Part type - text for TextParts*/
  kind: "text";
  /** Text content */
  text: string;
}
```

Field Name	Type	Required	Description
`kind`	`"text"` (literal)	Yes	Identifies this part as textual content.
`text`	`string`	Yes	The textual content of the part.
`metadata`	`Record<string, any>`	No	Optional metadata specific to this text part.

6.5.2. `FilePart` Object

For conveying file-based content.

```
/** Represents a File segment within parts.*/
export interface FilePart extends PartBase {
  /** Part type - file for FileParts */
  kind: "file";
  /** File content either as url or bytes */
  file: FileWithBytes | FileWithUri;
}
```

Field Name	Type	Required	Description
`kind`	`"file"` (literal)	Yes	Identifies this part as file content.
`file`	`FileWithBytes` | `FileWithUri`	Yes	Contains the file details and data/reference.
`metadata`	`Record<string, any>`	No	Optional metadata specific to this file part.

6.5.3. `DataPart` Object

For conveying structured JSON data. Useful for forms, parameters, or any machine-readable information.

```
/** Represents a structured data segment within a message part.
*/
export interface DataPart extends PartBase {
  /** Part type - data for DataParts */
  kind: "data";
  /** Structured data content
   */
  data: {
    [key: string]: any;
  };
}
```

Field Name	Type	Required	Description
`kind`	`"data"` (literal)	Yes	Identifies this part as structured data.
`data`	`Record<string, any>`	Yes	The structured JSON data payload (an object or an array).

Field Name	Type	Required	Description
`metadata`	`Record<string, any>`	No	Optional metadata specific to this data part (e.g., reference to a schema).

6.6.1 `FileWithBytes` Object

Represents the data for a file, used within a `FilePart`.

```
/** Define the variant where 'bytes' is present and 'uri' is absent */
export interface FileWithBytes extends FileBase {
  /** base64 encoded content of the file*/
  bytes: string;
  uri?: never;
}
```

Field Name	Type	Required	Description
`name`	`string`	No	Original filename (e.g., "report.pdf").
`mimeType`	`string`	No	Media Type (e.g., `image/png`). Strongly recommended.
`bytes`	`string`	Yes	Base64 encoded file content.

6.6.2 `FileWithUri` Object

Represents the URI for a file, used within a `FilePart`.

```
/** Define the variant where 'uri' is present and 'bytes' is absent  */
```

```ts
export interface FileWithUri extends FileBase {
  /** URL for the File content */
  uri: string;
  bytes?: never;
}
```

Field Name	Type	Required	Description
`name`	`string`	No	Original filename (e.g., "report.pdf").
`mimeType`	`string`	No	Media Type (e.g., `image/png`). Strongly recommended.
`uri`	`string`	Yes	URI (absolute URL strongly recommended) to file content. Accessibility is context-dependent.

6.7. `Artifact` Object

Represents a tangible output generated by the agent during a task. Artifacts are the results or products of the agent's work.

```ts
/** Represents an artifact generated for a task. */
export interface Artifact {
  /** Unique identifier for the artifact. */
  artifactId: string;
  /** Optional name for the artifact. */
  name?: string;
  /** Optional description for the artifact. */
  description?: string;
  /** Artifact parts. */
  parts: Part[];
  /** Extension metadata. */
  metadata?: {
    [key: string]: any;
  };
  /** The URIs of extensions that are present or contributed to this Artifact. */
  extensions?: string[];
```

```
}
```

Field Name	Type	Required	Description
`artifactId`	`string`	Yes	Identifier for the artifact generated by the agent.
`name`	`string`	No	Descriptive name for the artifact.
`description`	`string`	No	Human-readable description of the artifact.
`parts`	`Part[]`	Yes	Content of the artifact, as one or more `Part` objects. Must have at least one.
`metadata`	`Record<string, any>`	No	Arbitrary key-value metadata associated with the artifact.
`extensions`	`string[]`	No	A list of extension URIs that contributed to this artifact.

6.8. `PushNotificationConfig` Object

Configuration provided by the client to the server for sending asynchronous push notifications about task updates.

```
/**Configuration for setting up push notifications for task
updates. */
export interface PushNotificationConfig {
  /** Push Notification ID - created by server to support
multiple callbacks */
  id?: string;
```

```
  /** URL for sending the push notifications. */
  url: string;
  /** Token unique to this task/session. */
  token?: string;
  authentication?: PushNotificationAuthenticationInfo;
}
```

Field Name	Type	Required	Description
`url`	`string`	Yes	Absolute HTTPS webhook URL for the A2A Server to POST task updates to.
`token`	`string`	No	Optional client-generated opaque token for the client's webhook receiver to validate the notification (e.g., server includes it in an `X-A2A-Notification-Token` header).
`authentication`	`PushNotificationAuthenticationInfo`	No	Authentication details the A2A Server must use when calling the `url`. The client's webhook (receiver) defines these requirements.

6.9. `PushNotificationAuthenticationInfo` Object

A generic structure for specifying authentication requirements, typically used within `PushNotificationConfig` to describe how the A2A Server should authenticate to the client's webhook.

```
/** Defines authentication details for push notifications. */
export interface PushNotificationAuthenticationInfo {
```

```
  /** Supported authentication schemes - e.g. Basic, Bearer */
  schemes: string[];
  /** Optional credentials */
  credentials?: string;
}
```

Field Name	Type	Required	Description
`schemes`	`string[]`	Yes	Array of auth scheme names the A2A Server must use when calling the client's webhook (e.g., "Bearer", "ApiKey").
`credentials`	`string`	No	Optional static credentials or scheme-specific configuration info. **Handle with EXTREME CAUTION if secrets are involved.** Prefer server-side dynamic credential fetching where possible.

6.10. `TaskPushNotificationConfig` Object

Used as the `params` object for the `tasks/pushNotificationConfig/set` method and as the `result` object for the `tasks/pushNotificationConfig/get` method.

```
/**Parameters for setting or getting push notification
configuration for a task */
export interface TaskPushNotificationConfig {
  /** Task id. */
  taskId: string;
  /** Push notification configuration. */
  pushNotificationConfig: PushNotificationConfig;
}
```

Field Name	Type	Required	Description
`taskId`	`string`	Yes	The ID of the task to configure push notifications for, or retrieve configuration from.
`pushNotificationConfig`	`PushNotificationConfig`	Yes	The push notification configuration. For `set`, the desired config. For `get`, the current config (secrets MAY be omitted by server).

6.11. JSON-RPC Structures

A2A adheres to the standard JSON-RPC 2.0 structures for requests and responses.

6.11.1. `JSONRPCRequest` Object

All A2A method calls are encapsulated in a JSON-RPC Request object.

- `jsonrpc`: A String specifying the version of the JSON-RPC protocol. **MUST** be exactly `"2.0"`.
- `method`: A String containing the name of the method to be invoked (e.g., `"message/send"`, `"tasks/get"`).
- `params`: A Structured value that holds the parameter values to be used during the invocation of the method. This member **MAY** be omitted if the method expects no parameters. A2A methods typically use an `object` for `params`.
- `id`: An identifier established by the Client that **MUST** contain a String, Number, or `NULL` value if included. If it is not included it is assumed to be a notification. The value **SHOULD NOT** be `NULL` for requests expecting a response, and Numbers **SHOULD NOT** contain fractional parts. The Server **MUST** reply with the same value in the Response object if included. This member is used to correlate the context between the two

objects. A2A methods typically expect a response or stream, so `id` will usually be present and non-null.

6.11.2. `JSONRPCResponse` Object

Responses from the A2A Server are encapsulated in a JSON-RPC Response object.

- `jsonrpc`: A String specifying the version of the JSON-RPC protocol. **MUST** be exactly `"2.0"`.
- `id`: This member is **REQUIRED**. It **MUST** be the same as the value of the `id` member in the Request Object. If there was an error in detecting the `id` in the Request object (e.g. Parse error/Invalid Request), it **MUST** be `null`.
- **EITHER** `result`: This member is **REQUIRED** on success. This member **MUST NOT** exist if there was an error invoking the method. The value of this member is determined by the method invoked on the Server.
- **OR** `error`: This member is **REQUIRED** on failure. This member **MUST NOT** exist if there was no error triggered during invocation. The value of this member **MUST** be an `JSONRPCError` object.
- The members `result` and `error` are mutually exclusive: one **MUST** be present, and the other **MUST NOT**.

6.12. `JSONRPCError` Object

When a JSON-RPC call encounters an error, the Response Object will contain an `error` member with a value of this structure.

```
/**
 * Represents a JSON-RPC 2.0 Error object.
 * This is typically included in a JSONRPCErrorResponse when an
error occurs.
 */
export interface JSONRPCError {
  /**
   * A Number that indicates the error type that occurred.
   */
  code: number;

  /**
   * A String providing a short description of the error.
   */
  message: string;
```

```
  /**
   * A Primitive or Structured value that contains additional
information about the error.
   * This may be omitted.
   */
  data?: any;
}
```

Field Name	Type	Required	Description
`code`	`integer`	Yes	Integer error code. See Section 8 (Error Handling) for standard and A2A-specific codes.
`message`	`string`	Yes	Short, human-readable summary of the error.
`data`	`any`	No	Optional additional structured information about the error.

7. Protocol RPC Methods

All A2A RPC methods are invoked by the A2A Client by sending an HTTP POST request to the A2A Server's `url` (as specified in its `AgentCard`). The body of the HTTP POST request **MUST** be a `JSONRPCRequest` object, and the `Content-Type` header **MUST** be `application/json`.

The A2A Server's HTTP response body **MUST** be a `JSONRPCResponse` object (or, for streaming methods, an SSE stream where each event's data is a `JSONRPCResponse`). The `Content-Type` for JSON-RPC responses is `application/json`. For SSE streams, it is `text/event-stream`.

7.1. `message/send`

Sends a message to an agent to initiate a new interaction or to continue an existing one. This method is suitable for synchronous request/response interactions or when client-side polling (using `tasks/get`) is acceptable for monitoring longer-running tasks.

- **Request `params` type**: `MessageSendParams`
- **Response `result` type (on success)**: `Task` | `Message` (A message object or the current or final state of the task after processing the message).
- **Response `error` type (on failure)**: `JSONRPCError`.

7.1.1. `MessageSendParams` Object

```
/** Sent by the client to the agent as a request. May create,
continue or restart a task. */
export interface MessageSendParams {
  /** The message being sent to the server. */
  message: Message;
  /** Send message configuration. */
  configuration?: MessageSendConfiguration;
  /** Extension metadata. */
  metadata?: {
    [key: string]: any;
  };
}

/**Configuration for the send message request. */
export interface MessageSendConfiguration {
  /** Accepted output modalities by the client. */
  acceptedOutputModes: string[];
  /** Number of recent messages to be retrieved. */
  historyLength?: number;
  /** Where the server should send notifications when
disconnected. */
  pushNotificationConfig?: PushNotificationConfig;
  /** If the server should treat the client as a blocking
request. */
  blocking?: boolean;
}
```

Field Name	Type	Required	Description
`message`	`Message`	Yes	The message content to send. `Message.role` is typically `"user"`.
`configuration`	`MessageSendConfiguration`	No	Optional: additional message configuration
`metadata`	`Record<string, any>`	No	Request-specific metadata.

7.2. `message/stream`

Sends a message to an agent to initiate/continue a task AND subscribes the client to real-time updates for that task via Server-Sent Events (SSE). This method requires the server to have `AgentCard.capabilities.streaming: true`.

- **Request `params` type**: `MessageSendParams` (same as `message/send`).
- **Response (on successful subscription)**:
  - HTTP Status: `200 OK`.
  - HTTP `Content-Type`: `text/event-stream`.
  - HTTP Body: A stream of Server-Sent Events. Each SSE `data` field contains a `SendStreamingMessageResponse` JSON object.
- **Response (on initial subscription failure)**:
  - Standard HTTP error code (e.g., 4xx, 5xx).
  - The HTTP body MAY contain a standard `JSONRPCResponse` with an `error` object detailing the failure.

7.2.1. `SendStreamingMessageResponse` Object

This is the structure of the JSON object found in the `data` field of each Server-Sent Event sent by the server for a `message/stream` request or `tasks/resubscribe` request.

```ts
/**
 * JSON-RPC response model for the 'message/stream' method.
 */
export type SendStreamingMessageResponse =
  | SendStreamingMessageSuccessResponse
  | JSONRPCErrorResponse;

/**
 * JSON-RPC success response model for the 'message/stream'
method.
 */
export interface SendStreamingMessageSuccessResponse
  extends JSONRPCSuccessResponse {
  result: Message | Task | TaskStatusUpdateEvent |
TaskArtifactUpdateEvent;
}
```

Field Name	Type	Required	Description
`jsonrpc`	`"2.0"` (literal)	Yes	JSON-RPC version string.
`id`	`string` | `number`	Yes	Matches the `id` from the originating `message/stream` or `tasks/resubscribe` request.
`result`	**Either** `Message` **OR** `Task` **OR** `TaskStatusUpdateEvent` **OR** `TaskArtifactUpdateEvent`	Yes	The event payload

7.2.2. `TaskStatusUpdateEvent` Object

Carries information about a change in the task's status during streaming. This is one of the possible `result` types in a `SendStreamingMessageSuccessResponse`.

```
/** Sent by server during sendStream or subscribe requests */
export interface TaskStatusUpdateEvent {
  /** Task id */
  taskId: string;
  /** The context the task is associated with */
  contextId: string;
  /** Event type */
  kind: "status-update";
  /** Current status of the task */
  status: TaskStatus;
  /** Indicates the end of the event stream */
  final: boolean;
  /** Extension metadata. */
  metadata?: {
    [key: string]: any;
  };
}
```

Field Name	Type	Required	Default	Description
`taskId`	`string`	Yes		Task ID being updated
`contextId`	`string`	Yes		Context ID the task is associated with
`kind`	`string`, literal	Yes	`status-update`	Type discriminator literal value
`status`	`TaskStatus`	Yes		The new `TaskStatus` object.
`final`	`boolean`	No	`false`	If `true`, indicates thi

Field Name	Type	Required	Default	Description
				is the terminal status update for the current stream cycle The server typically closes the SSE connection after this.
`metadata`	`Record<string, any>`	No	`undefined`	Event-specific metadata.

7.2.3. `TaskArtifactUpdateEvent` Object

Carries a new or updated artifact (or a chunk of an artifact) generated by the task during streaming. This is one of the possible `result` types in a `SendTaskStreamingResponse`.

```ts
/** Sent by server during sendStream or subscribe requests */
export interface TaskArtifactUpdateEvent {
  /** Task id */
  taskId: string;
  /** The context the task is associated with */
  contextId: string;
  /** Event type */
  kind: "artifact-update";
  /** Generated artifact */
  artifact: Artifact;
  /** Indicates if this artifact appends to a previous one */
  append?: boolean;
  /** Indicates if this is the last chunk of the artifact */
  lastChunk?: boolean;
  /** Extension metadata. */
  metadata?: {
    [key: string]: any;
  };
```

```
}
```

Field Name	Type	Required	Default	Description
`taskId`	`string`	Yes		Task ID associated with the generated artifact part
`contextId`	`string`	Yes		Context ID the task is associated with
`kind`	`string`, literal	Yes	`artifact-update`	Type discriminato literal value
`artifact`	`Artifact`	Yes		The `Artifact` data. Could be a complete artifact or ar incremental chunk.
`append`	`boolean`	No	`false`	`true` means append parts to artifact; `false` (default) means replace.
`lastChunk`	`boolean`	No	`false`	`true` indicates thi

Field Name	Type	Required	Default	Description
				is the final update for the artifact.
`metadata`	`Record<string, any>`	No	`undefined`	Event-specific metadata.

7.3. `tasks/get`

Retrieves the current state (including status, artifacts, and optionally history) of a previously initiated task. This is typically used for polling the status of a task initiated with `message/send`, or for fetching the final state of a task after being notified via a push notification or after an SSE stream has ended.

- **Request `params` type:** `TaskQueryParams`
- **Response `result` type (on success):** `Task` (A snapshot of the task's current state).
- **Response `error` type (on failure):** `JSONRPCError` (e.g., if the task ID is not found, see `TaskNotFoundError`).

7.3.1. `TaskQueryParams` Object

```
/** Parameters for querying a task, including optional history
length. */
export interface TaskQueryParams extends TaskIdParams {
  /** Number of recent messages to be retrieved. */
  historyLength?: number;
}
```

Field Name	Type	Required	Description
`id`	`string`	Yes	The ID of the task whose current state is to be retrieved.

Field Name	Type	Required	Description
`historyLength`	`integer`	No	If positive, requests the server to include up to `N` recent messages in `Task.history`.
`metadata`	`Record<string, any>`	No	Request-specific metadata.

7.4. `tasks/cancel`

Requests the cancellation of an ongoing task. The server will attempt to cancel the task, but success is not guaranteed (e.g., the task might have already completed or failed, or cancellation might not be supported at its current stage).

- **Request `params` type**: `TaskIdParams`
- **Response `result` type (on success)**: `Task` (The state of the task after the cancellation attempt. Ideally, `Task.status.state` will be `"canceled"` if successful).
- **Response `error` type (on failure)**: `JSONRPCError` (e.g., `TaskNotFoundError`, `TaskNotCancelableError`).

7.4.1. `TaskIdParams` Object (for `tasks/cancel` and `tasks/pushNotificationConfig/get`)

A simple object containing just the task ID and optional metadata.

```
/** Parameters containing only a task ID, used for simple task
operations. */
export interface TaskIdParams {
  /** Task id. */
  id: string;
  metadata?: {
    [key: string]: any;
  };
}
```

Field Name	Type	Required	Description
`id`	`string`	Yes	The ID of the task.
`metadata`	`Record<string, any>`	No	Request-specific metadata.

7.5. `tasks/pushNotificationConfig/set`

Sets or updates the push notification configuration for a specified task. This allows the client to tell the server where and how to send asynchronous updates for the task. Requires the server to have `AgentCard.capabilities.pushNotifications: true`.

- **Request `params` type**: `TaskPushNotificationConfig`
- **Response `result` type (on success)**: `TaskPushNotificationConfig` (Confirms the configuration that was set. The server MAY omit or mask any sensitive details like secrets from the `authentication.credentials` field in the response).
- **Response `error` type (on failure)**: `JSONRPCError` (e.g., `PushNotificationNotSupportedError`, `TaskNotFoundError`, errors related to invalid `PushNotificationConfig`).

7.6. `tasks/pushNotificationConfig/get`

Retrieves the current push notification configuration for a specified task. Requires the server to have `AgentCard.capabilities.pushNotifications: true`.

- **Request `params` type**: `TaskIdParams`
- **Response `result` type (on success)**: `TaskPushNotificationConfig` (The current push notification configuration for the task. Server may return an error if no push notification configuration is associated with the task).
- **Response `error` type (on failure)**: `JSONRPCError` (e.g., `PushNotificationNotSupportedError`, `TaskNotFoundError`).

7.7. `tasks/resubscribe`

Allows a client to reconnect to an SSE stream for an ongoing task after a previous connection (from `message/stream` or an earlier `tasks/resubscribe`) was interrupted. Requires the server to have `AgentCard.capabilities.streaming: true`.

The purpose is to resume receiving *subsequent* updates. The server's behavior regarding events missed during the disconnection period (e.g., whether it attempts to backfill some missed events or only sends new ones from the point of resubscription) is implementation-dependent and not strictly defined by this specification.

- **Request `params` type**: `TaskIdParams`
- **Response (on successful resubscription)**:
  - HTTP Status: `200 OK`.
  - HTTP `Content-Type`: `text/event-stream`.
  - HTTP Body: A stream of Server-Sent Events, identical in format to `message/stream`, carrying *subsequent* `SendStreamingMessageResponse` events for the task.
- **Response (on resubscription failure)**:
  - Standard HTTP error code (e.g., 4xx, 5xx).
  - The HTTP body MAY contain a standard `JSONRPCResponse` with an `error` object. Failures can occur if the task is no longer active, doesn't exist, or streaming is not supported/enabled for it.

7.8. `agent/authenticatedExtendedCard`

Retrieves a potentially more detailed version of the Agent Card after the client has authenticated. This endpoint is available only if `AgentCard.supportsAuthenticatedExtendedCard` is `true`. This is an HTTP GET endpoint, not a JSON-RPC method.

- **Endpoint URL**: `{AgentCard.url}/../agent/authenticatedExtendedCard` (relative to the base URL specified in the public Agent Card).
- **HTTP Method**: `GET`

- **Authentication**: The client **MUST** authenticate the request using one of the schemes declared in the public `AgentCard.securitySchemes` and `AgentCard.security` fields.
- **Request `params`**: None (HTTP GET request).
- **Response `result` type (on success)**: `AgentCard` (A complete Agent Card object, which may contain additional details or skills not present in the public card).
- **Response `error` type (on failure)**: Standard HTTP error codes.
  - `401 Unauthorized`: Authentication failed (missing or invalid credentials). The server **SHOULD** include a `WWW-Authenticate` header.
  - `403 Forbidden`: Authentication succeeded, but the client/user is not authorized to access the extended card.
  - `404 Not Found`: The `supportsAuthenticatedExtendedCard` capability is declared, but the server has not implemented this endpoint at the specified path.
  - `5xx Server Error`: An internal server error occurred.

Clients retrieving this authenticated card **SHOULD** replace their cached public Agent Card with the content received from this endpoint for the duration of their authenticated session or until the card's version changes.

7.8.1. `AuthenticatedExtendedCardParams` Object

This endpoint does not use JSON-RPC `params`. Any parameters would be included as HTTP query parameters if needed (though none are defined by the standard).

7.8.2. `AuthenticatedExtendedCardResponse` Object

The successful response body is a JSON object conforming to the `AgentCard` interface.

```
```

8. Error Handling

A2A uses standard JSON-RPC 2.0 error codes and structure for reporting errors. Errors are returned in the `error` member of the `JSONRPCErrorResponse` object. See `JSONRPCError` Object definition.

8.1. Standard JSON-RPC Errors

These are standard codes defined by the JSON-RPC 2.0 specification.

Code	JSON-RPC Spec Meaning	Typical A2A `message`	Description
`-32700`	Parse error	Invalid JSON payload	Server received JSON that was not well-formed.
`-32600`	Invalid Request	Invalid JSON-RPC Request	The JSON payload was valid JSON, but not a valid JSON-RPC Request object.
`-32601`	Method not found	Method not found	The requested A2A RPC `method` (e.g., `"tasks/foo"`) does not exist or is not supported.
`-32602`	Invalid params	Invalid method parameters	The `params` provided for the method are invalid (e.g., wrong type, missing required field).
`-32603`	Internal error	Internal server error	An unexpected error occurred on the server during processing.

Code	JSON-RPC Spec Meaning	Typical A2A `message`	Description
`-32000` to `-32099`	Server error	*(Server-defined)*	Reserved for implementation-defined server-errors. A2A-specific errors use this range.

8.2. A2A-Specific Errors

These are custom error codes defined within the JSON-RPC server error range (`-32000` to `-32099`) to provide more specific feedback about A2A-related issues. Servers **SHOULD** use these codes where applicable.

Code	Error Name (Conceptual)	Typical `message` string	Description
`-32001`	`TaskNotFoundError`	Task not found	The specified task `id` does not correspond to an existing or active task. It might be invalid, expired, or already completed and purged.

Code	Error Name (Conceptual)	Typical `message` string	Description
`-32002`	`TaskNotCancelableError`	Task cannot be canceled	An attempt was made to cancel a task that is not in a cancelable state (e.g., it has already reached a terminal state like `completed`, `failed`, or `canceled`).
`-32003`	`PushNotificationNotSupportedError`	Push Notification is not supported	Client attempted to use push notification features (e.g., `tasks/pushNotificationConfig/set`) but the server agent does not support them (i.e., `AgentCard.capabilities.pushNotifications` is `false`).
`-32004`	`UnsupportedOperationError`	This operation is not supported	The requested operation or a specific aspect of it (perhaps implied by parameters) is not supported by this server agent implementation. Broader than just method not found.

Code	Error Name (Conceptual)	Typical `message` string	Description
`-32005`	`ContentTypeNotSupportedError`	Incompatible content types	A Media Type provided in the request's `message.parts` (or implied for an artifact) is not supported by the agent or the specific skill being invoked.
`-32006`	`InvalidAgentResponseError`	Invalid agent response type	Agent generated an invalid response for the requested method

Servers MAY define additional error codes within the `-32000` to `-32099` range for more specific scenarios not covered above, but they **SHOULD** document these clearly. The `data` field of the `JSONRPCError` object can be used to provide more structured details for any error.

9. Common Workflows & Examples

This section provides illustrative JSON examples of common A2A interactions. Timestamps, context IDs, and request/response IDs are for demonstration purposes. For brevity, some optional fields might be omitted if not central to the example.

9.1. Fetching Authenticated Extended Agent Card

Scenario: A client discovers a public Agent Card indicating support for an authenticated extended card and wants to retrieve the full details.

1. **Client fetches the public Agent Card:**

```
GET https://example.com/.well-known/agent.json
```

Server responds with the public Agent Card (like the example in Section 5.6), including `supportsAuthenticatedExtendedCard: true` (at the root level) and `securitySchemes`.

1. **Client identifies required authentication from the public card.**
2. **Client obtains necessary credentials out-of-band (e.g., performs OAuth 2.0 flow with Google, resulting in an access token).**
3. **Client fetches the authenticated extended Agent Card:**

```
GET https://example.com/a2a/agent/authenticatedExtendedCard
Authorization: Bearer <obtained_access_token>
```

1. **Server authenticates and authorizes the request.**
2. **Server responds with the full Agent Card:**

9.2. Basic Execution (Synchronous / Polling Style)

Scenario: Client asks a simple question, and the agent responds quickly with a task

1. **Client sends a message using `message/send`:**

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "tell me a joke"
        }
      ],
      "messageId": "9229e770-767c-417b-a0b0-f0741243c589"
    },
    "metadata": {}
  }
}
```

1. **Server processes the request, creates a task and responds (task completes quickly)**


```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "id": "363422be-b0f9-4692-a24d-278670e7c7f1",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "status": {
      "state": "completed"
    },
    "artifacts": [
      {
        "artifactId": "9b6934dd-37e3-4eb1-8766-962efaab63a1",
        "name": "joke",
        "parts": [
          {
            "kind": "text",
            "text": "Why did the chicken cross the road? To get
to the other side!"
          }
        ]
      }
    ],
    "history": [
      {
        "role": "user",
        "parts": [
          {
            "kind": "text",
            "text": "tell me a joke"
          }
        ],
        "messageId": "9229e770-767c-417b-a0b0-f0741243c589",
        "taskId": "363422be-b0f9-4692-a24d-278670e7c7f1",
        "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4"
      }
    ],
    "kind": "task",
    "metadata": {}
  }
}
```


Scenario: Client asks a simple question, and the agent responds quickly without a task

1. **Client sends a message using `message/send`:**


```
{
```

```
  "jsonrpc": "2.0",
  "id": 1,
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "tell me a joke"
        }
      ],
      "messageId": "9229e770-767c-417b-a0b0-f0741243c589"
    },
    "metadata": {}
  }
}
```

1. **Server processes the request, responds quickly without a task**

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "messageId": "363422be-b0f9-4692-a24d-278670e7c7f1",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "parts": [
      {
        "kind": "text",
        "text": "Why did the chicken cross the road? To get to
the other side!"
      }
    ],
    "kind": "message",
    "metadata": {}
  }
}
```

If the task were longer-running, the server might initially respond with `status.state: "working"`. The client would then periodically call `tasks/get` with `params: {"id": "363422be-b0f9-4692-a24d-278670e7c7f1"}` until the task reaches a terminal state.

9.3. Streaming Task Execution (SSE)

Scenario: Client asks the agent to write a long paper describing an attached picture.

1. **Client sends a message and subscribes using `message/stream`:**

```
{
  "method": "message/stream",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "write a long paper describing the attached
pictures"
        },
        {
          "kind": "file",
          "file": {
            "mimeType": "image/png",
            "data": "<base64-encoded-content>"
          }
        }
      ],
      "messageId": "bbb7dee1-cf5c-4683-8a6f-4114529da5eb"
    },
    "metadata": {}
  }
}
```

1. **Server responds with HTTP 200 OK, `Content-Type: text/event-stream`, and starts sending SSE events:**

Event 1: Task status update - working

```
data: {
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "id": "225d6247-06ba-4cda-a08b-33ae35c8dcfa",
    "contextId": "05217e44-7e9f-473e-ab4f-2c2dde50a2b1",
    "status": {
      "state": "submitted",
      "timestamp":"2025-04-02T16:59:25.331844"
    },
    "history": [
      {
        "role": "user",
        "parts": [
          {
            "kind": "text",
            "text": "write a long paper describing the attached
pictures"
          },
```

```
          {
            "kind": "file",
            "file": {
              "mimeType": "image/png",
              "data": "<base64-encoded-content>"
            }
          }
        ],
        "messageId": "bbb7dee1-cf5c-4683-8a6f-4114529da5eb",
        "taskId": "225d6247-06ba-4cda-a08b-33ae35c8dcfa",
        "contextId": "05217e44-7e9f-473e-ab4f-2c2dde50a2b1"
      }
    ],
    "kind": "task",
    "metadata": {}
  }
}

data: {
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "taskId": "225d6247-06ba-4cda-a08b-33ae35c8dcfa",
    "contextId": "05217e44-7e9f-473e-ab4f-2c2dde50a2b1",
    "artifact": {
      "artifactId": "9b6934dd-37e3-4eb1-8766-962efaab63a1",
      "parts": [
        {"type":"text", "text": "<section 1...>"}
      ]
    },
    "append": false,
    "lastChunk": false,
    "kind":"artifact-update"
  }
}

data: {
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "taskId": "225d6247-06ba-4cda-a08b-33ae35c8dcfa",
    "contextId": "05217e44-7e9f-473e-ab4f-2c2dde50a2b1",
    "artifact": {
      "artifactId": "9b6934dd-37e3-4eb1-8766-962efaab63a1",
      "parts": [
        {"type":"text", "text": "<section 2...>"}
      ],
    },
    "append": true,
    "lastChunk": false,
    "kind":"artifact-update"
  }
}
```

```
data: {
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "taskId": "225d6247-06ba-4cda-a08b-33ae35c8dcfa",
    "contextId": "05217e44-7e9f-473e-ab4f-2c2dde50a2b1",
    "artifact": {
      "artifactId": "9b6934dd-37e3-4eb1-8766-962efaab63a1",
      "parts": [
        {"type":"text", "text": "<section 3...>"}
      ]
    },
    "append": true,
    "lastChunk": true,
    "kind":"artifact-update"
  }
}

data: {
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "taskId": "225d6247-06ba-4cda-a08b-33ae35c8dcfa",
    "contextId": "05217e44-7e9f-473e-ab4f-2c2dde50a2b1",
    "status": {
      "state": "completed",
      "timestamp":"2025-04-02T16:59:35.331844"
    },
    "final": true,
    "kind":"status-update"
  }
}
```

(Server closes the SSE connection after the `final:true` event).

9.4. Multi-Turn Interaction (Input Required)

Scenario: Client wants to book a flight, and the agent needs more information.

1. **Client sends a message using `message/send`:**

```
{
  "jsonrpc": "2.0",
  "id": "req-003",
  "method": "message/send",
  "params": {
    "message": {
```

```
      "role": "user",
      "parts": [{ "kind": "text", "text": "I'd like to book a
flight." }]
    },
    "messageId": "c53ba666-3f97-433c-a87b-6084276babe2"
  }
}
```

1. **Server responds, task state is `input-required`:**

```
{
  "jsonrpc": "2.0",
  "id": "req-003",
  "result": {
    "id": "3f36680c-7f37-4a5f-945e-d78981fafd36",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "status": {
      "state": "input-required",
      "message": {
        "role": "agent",
        "parts": [
          {
            "kind": "text",
            "text": "Sure, I can help with that! Where would
you like to fly to, and from where? Also, what are your
preferred travel dates?"
          }
        ],
        "messageId": "c2e1b2dd-f200-4b04-bc22-1b0c65a1aad2",
        "taskId": "3f36680c-7f37-4a5f-945e-d78981fafd36",
        "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4"
      },
      "timestamp": "2024-03-15T10:10:00Z"
    },
    "history": [
      {
        "role": "user",
        "parts": [
          {
            "kind": "text",
            "text": "I'd like to book a flight."
          }
        ],
        "messageId": "c53ba666-3f97-433c-a87b-6084276babe2",
        "taskId": "3f36680c-7f37-4a5f-945e-d78981fafd36",
        "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4"
      }
    ],
    "kind": "task"
  }
}
```

1. **Client `message/send` (providing the requested input, using the *same* task ID):**

```
{
  "jsonrpc": "2.0",
  "id": "req-004",
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "I want to fly from New York (JFK) to London
(LHR) around October 10th, returning October 17th."
        }
      ],
      "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
      "taskId": "3f36680c-7f37-4a5f-945e-d78981fafd36",
      "messageId": "0db1d6c4-3976-40ed-b9b8-0043ea7a03d3"
    },
    "configuration": {
      "blocking": true
    }
  }
}
```

1. **Server processes the new input and responds (e.g., task completed or more input needed):**

```
{
  "jsonrpc": "2.0",
  "id": "req-004",
  "result": {
    "id": "3f36680c-7f37-4a5f-945e-d78981fafd36",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "status": {
      "state": "completed",
      "message": {
        "role": "agent",
        "parts": [
          {
            "kind": "text",
            "text": "Okay, I've found a flight for you.
Confirmation XYZ123. Details are in the artifact."
          }
        ]
      }
    },
    "artifacts": [
      {
```

```
        "artifactId": "9b6934dd-37e3-4eb1-8766-962efaab63a1",
        "name": "FlightItinerary.json",
        "parts": [
          {
            "kind": "data",
            "data": {
              "confirmationId": "XYZ123",
              "from": "JFK",
              "to": "LHR",
              "departure": "2024-10-10T18:00:00Z",
              "arrival": "2024-10-11T06:00:00Z",
              "returnDeparture": "..."
            }
          }
        ]
      }
    ],
    "history": [
      {
        "role": "user",
        "parts": [
          {
            "kind": "text",
            "text": "I'd like to book a flight."
          }
        ],
        "messageId": "c53ba666-3f97-433c-a87b-6084276babe2",
        "taskId": "3f36680c-7f37-4a5f-945e-d78981fafd36",
        "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4"
      },
      {
        "role": "agent",
        "parts": [
          {
            "kind": "text",
            "text": "Sure, I can help with that! Where would
you like to fly to, and from where? Also, what are your
preferred travel dates?"
          }
        ],
        "messageId": "c2e1b2dd-f200-4b04-bc22-1b0c65a1aad2",
        "taskId": "3f36680c-7f37-4a5f-945e-d78981fafd36",
        "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4"
      },
      {
        "role": "user",
        "parts": [
          {
            "kind": "text",
            "text": "I want to fly from New York (JFK) to
London (LHR) around October 10th, returning October 17th."
          }
        ],
```

```
        "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
        "taskId": "3f36680c-7f37-4a5f-945e-d78981fafd36",
        "messageId": "0db1d6c4-3976-40ed-b9b8-0043ea7a03d3"
      }
    ],
    "kind": "task",
    "metadata": {}
  }
}
```

9.5. Push Notification Setup and Usage

Scenario: Client requests a long-running report generation and wants to be notified via webhook when it's done.

1. **Client `message/send` with `pushNotification` config:**

```
{
  "jsonrpc": "2.0",
  "id": "req-005",
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "Generate the Q1 sales report. This usually
takes a while. Notify me when it's ready."
        }
      ],
      "messageId": "6dbc13b5-bd57-4c2b-b503-24e381b6c8d6"
    },
    "configuration": {
      "pushNotificationConfig": {
        "url": "https://client.example.com/webhook/a2a-
notifications",
        "token": "secure-client-token-for-task-aaa",
        "authentication": {
          "schemes": ["Bearer"]
          // Assuming server knows how to get a Bearer token
for this webhook audience,
          // or this implies the webhook is public/uses the
'token' for auth.
          // 'credentials' could provide more specifics if
needed by the server.
        }
      }
    }
  }
```

```
}
```

1. **Server acknowledges the task (e.g., status `submitted` or `working`):**

```
{
  "jsonrpc": "2.0",
  "id": "req-005",
  "result": {
    "id": "43667960-d455-4453-b0cf-1bae4955270d",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "status": { "state": "submitted", "timestamp": "2024-03-
15T11:00:00Z" }
    // ... other fields ...
  }
}
```

1. **(Later) A2A Server completes the task and POSTs a notification to `https://client.example.com/webhook/a2a-notifications`:**
2. **HTTP Headers might include:**
   - `Authorization: Bearer <server_jwt_for_webhook_audience>` (if server authenticates to webhook)
   - `Content-Type: application/json`
   - `X-A2A-Notification-Token: secure-client-token-for-task-aaa`
3. **HTTP Body (Task object is sent as JSON payload):**

```
{
  "id": "43667960-d455-4453-b0cf-1bae4955270d",
  "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
  "status": { "state": "completed", "timestamp": "2024-03-
15T18:30:00Z" },
  "kind": "task"
  // ... other fields ...
}
```

1. **Client's Webhook Service:**
2. Receives the POST.
3. Validates the `Authorization` header (if applicable).
4. Validates the `X-A2A-Notification-Token`.
5. Internally processes the notification (e.g., updates application state, notifies end user).

9.6. File Exchange (Upload and Download)

Scenario: Client sends an image for analysis, and the agent returns a modified image.

1. **Client `message/send` with a `FilePart` (uploading image bytes):**

```
{
  "jsonrpc": "2.0",
  "id": "req-007",
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "Analyze this image and highlight any faces."
        },
        {
          "kind": "file",
          "file": {
            "name": "input_image.png",
            "mimeType": "image/png",
            "bytes": "iVBORw0KGgoAAAANSUhEUgAAAUA..." //
Base64 encoded image data
          }
        }
      ],
      "messageId": "6dbc13b5-bd57-4c2b-b503-24e381b6c8d6"
    }
  }
}
```

1. **Server processes the image and responds with a `FilePart` in an artifact (e.g., providing a URI to the modified image):**

```
{
  "jsonrpc": "2.0",
  "id": "req-007",
  "result": {
    "id": "43667960-d455-4453-b0cf-1bae4955270d",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "status": { "state": "completed", "timestamp": "2024-03-
15T12:05:00Z" },
    "artifacts": [
      {
        "artifactId": "9b6934dd-37e3-4eb1-8766-962efaab63a1",
        "name": "processed_image_with_faces.png",
        "parts": [
```

```
          {
            "kind": "file",
            "file": {
              "name": "output.png",
              "mimeType": "image/png",
              // Server might provide a URI to a temporary
storage location
              "uri":
"https://storage.example.com/processed/task-bbb/output.png?
token=xyz"
              // Or, alternatively, it could return bytes
directly:
              // "bytes": "ASEDGhw0KGgoAAAANSUhEUgAA..."
            }
          }
        ]
      }
    ],
    "kind": "task"
  }
}
```

9.7. Structured Data Exchange (Requesting and Providing JSON)

Scenario: Client asks for a list of open support tickets in a specific JSON format.

1. **Client `message/send`, `Part.metadata` hints at desired output schema/Media Type:** *(Note: A2A doesn't formally standardize schema negotiation in v0.2.0, but `metadata` can be used for such hints by convention between client/server).*

```
{
  "jsonrpc": "2.0",
  "id": 9,
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "Show me a list of my open IT tickets",
          "metadata": {
            "mimeType": "application/json",
            "schema": {
              "type": "array",
              "items": {
```

```
                "type": "object",
                "properties": {
                  "ticketNumber": { "type": "string" },
                  "description": { "type": "string" }
                }
              }
            }
          }
        }
      ],
      "messageId": "85b26db5-ffbb-4278-a5da-a7b09dea1b47"
    },
    "metadata": {}
  }
}
```

1. **Server responds with structured JSON data:**

```
{
  "jsonrpc": "2.0",
  "id": 9,
  "result": {
    "id": "d8c6243f-5f7a-4f6f-821d-957ce51e856c",
    "contextId": "c295ea44-7543-4f78-b524-7a38915ad6e4",
    "status": {
      "state": "completed",
      "timestamp": "2025-04-17T17:47:09.680794"
    },
    "artifacts": [
      {
        "artifactId": "c5e0382f-b57f-4da7-87d8-b85171fad17c",
        "parts": [
          {
            "kind": "text",
            "text": "
[{\"ticketNumber\":\"REQ12312\",\"description\":\"request for
VPN access\"},
{\"ticketNumber\":\"REQ23422\",\"description\":\"Add to DL -
team-gcp-onboarding\"}]"
          }
        ]
      }
    ],
    "kind": "task"
  }
}
```

These examples illustrate the flexibility of A2A in handling various interaction patterns and data types. Implementers should refer to the detailed object definitions for all fields and constraints.

10. Appendices

10.1. Relationship to MCP (Model Context Protocol)

A2A and MCP are complementary protocols designed for different aspects of agentic systems:

- **Model Context Protocol (MCP):** Focuses on standardizing how AI models and agents connect to and interact with **tools, APIs, data sources, and other external resources.** It defines structured ways to describe tool capabilities (like function calling in LLMs), pass inputs, and receive structured outputs. Think of MCP as the "how-to" for an agent to *use* a specific capability or access a resource.
- **Agent2Agent Protocol (A2A):** Focuses on standardizing how independent, often opaque, **AI agents communicate and collaborate with each other as peers.** A2A provides an application-level protocol for agents to discover each other, negotiate interaction modalities, manage shared tasks, and exchange conversational context or complex results. It's about how agents *partner* or *delegate* work.

How they work together: An A2A Client agent might request an A2A Server agent to perform a complex task. The Server agent, in turn, might use MCP to interact with several underlying tools, APIs, or data sources to gather information or perform actions necessary to fulfill the A2A task.

For a more detailed comparison, see the A2A and MCP guide.

10.2. Security Considerations Summary

Security is a paramount concern in A2A. Key considerations include:

- **Transport Security:** Always use HTTPS with strong TLS configurations in production environments.
- **Authentication:**
  - Handled via standard HTTP mechanisms (e.g., `Authorization` header with Bearer tokens, API keys).
  - Requirements are declared in the `AgentCard`.
  - Credentials MUST be obtained out-of-band by the client.
  - A2A Servers MUST authenticate every request.

- **Authorization:**
  - A server-side responsibility based on the authenticated identity.
  - Implement the principle of least privilege.
  - Can be granular, based on skills, actions, or data.
- **Push Notification Security:**
  - Webhook URL validation (by the A2A Server sending notifications) is crucial to prevent SSRF.
  - Authentication of the A2A Server to the client's webhook is essential.
  - Authentication of the notification by the client's webhook receiver (verifying it came from the legitimate A2A Server and is relevant) is critical.
  - See the Streaming & Asynchronous Operations guide for detailed push notification security.
- **Input Validation:** Servers MUST rigorously validate all RPC parameters and the content/structure of data in `Message` and `Artifact` parts to prevent injection attacks or processing errors.
- **Resource Management:** Implement rate limiting, concurrency controls, and resource limits to protect agents from abuse or overload.
- **Data Privacy:** Adhere to all applicable privacy regulations for data exchanged in `Message` and `Artifact` parts. Minimize sensitive data transfer.

For a comprehensive discussion, refer to the Enterprise-Ready Features guide.