OpenAV Cloud-to-Cloud REST API Guidelines

Document Version: 1.1
Prepared by: OpenAVCloud Technical Working Group
Date: August 5th, 2026

Table of Contents

Document History and Change Log

This section captures the history of changes made to this document.

Version	Date	Reason for Change
0.2	2026-04-03	Draft
0.3	2026-04-13	Update per feedback
1.0	2026-06-02	Initial release
1.1	2026-08-05	Official release

1. Purpose

The goal of this API is to enable cloud-to-cloud interoperability between OpenAV-compatible systems. The API is intended to expose device and system capabilities in a capability-centric, standards-aligned way that supports scalable integration across multiple vendors and cloud platforms.

This guideline builds on industry best practices, with an emphasis on:

1.1 Interoperability
1.2 Extensibility
1.3 Secure, standards-based access
1.4 Event-driven integration

2. Design Principles

2.1 Architectural Style

2.1.1 RESTful API design
2.1.2 Stateless interactions for scalability and reliability
2.1.3 Resource-oriented URLs are preferred (`e.g., /v1/devices, /v1/devices/{deviceId}/capabilities`)
2.1.4 JSON for request and response payloads

2.2 Capability-Centric Model

2.2.1 The API is capability-centric, not strictly device-centric, as a means of defining a functionality applicable across one or more devices.
2.2.2 Devices expose one or more capabilities
2.2.3 Capabilities may be:
  2.2.3.1 Standardized across manufacturers (e.g. `audio-mute`)
  2.2.3.2 Manufacturer-specific extensions, but namespaced to avoid collisions (e.g. `com.manufacturer.capability-name`)
2.2.4 To reduce or eliminate client-coded assumptions for what interactions are possible with devices (and their corresponding firmware versions), the following principles must be upheld:
  2.2.4.1 API clients must be able to dynamically determine what devices are capable of at runtime using HATEOAS-inspired discoverability
  2.2.4.2 Every capability must have a means of indicating the superset of all related APIs pertaining to it so clients can establish the relationship between any given capability and its corresponding APIs either during development, or at application runtime.
2.2.5 See Section 12 for more details

2.3 Standards Alignment

2.3.1 OAuth 2.0 for authentication
2.3.2 OData JSON format used for request/response batching

3. API Versioning

3.1 API versioning is included in the URL path.

3.1.1 Example:

```
/v1/...
```

3.1.2 Changing the top-level API version indicates a major breaking change affecting the overall API contract.

3.1.3 This type of version change is used for broad or cross-cutting changes, such as:

3.1.3.1 Changes affecting multiple endpoints

3.1.3.2 Shared request/response structure changes

3.1.3.3 Authentication or authorization model changes

3.1.3.4 Common behavioral changes across the API

3.1.4 Example:

```
/v1/... -> /v2/...
```

3.1.5 Backward compatibility should be maintained where possible while enabling iterative improvements.

3.1.6 Consumers should ignore any undocumented response fields to allow future expansion.

3.2 Endpoint-Specific Versioning

3.2.1 Individual endpoints may also be versioned independently when a breaking change affects only a single API or isolated feature.

3.2.2 In these cases, the overall API version remains unchanged, and only the endpoint name is versioned.

3.2.3 Example:

```
/v1/devices/{deviceId}/audio-channels
/v1/devices/{deviceId}/audio-channels-v2
```

3.2.4 Endpoint-specific versioning should be used when:

3.2.4.1 The breaking change is isolated to a single endpoint

3.2.4.2 Other APIs remain fully backward compatible

3.2.4.3 A full API version bump would be unnecessary

3.2.5 A top-level API version bump (for example, `/v2/...`) should be used when the breaking change impacts multiple APIs or the platform contract as a whole.

4. HTTP Methods & Idempotency

4.1 Method Usage

4.1.1 GET – Retrieve resources
4.1.2 POST – Create resources
4.1.3 PUT – Update resources
4.1.4 DELETE – Delete resources

4.2 Idempotency

4.2.1 Idempotency rules for device-related operations is preferred when possible, but is not strictly required.
4.2.2 For example, operations such as triggering a firmware update may not be idempotent by nature, as the outcome depends on the current state of the device at the time of execution.

5. Pagination

5.1.1 Pagination shall be possible for limiting the number of records returned.
5.1.2 It must be possible to seek through pages of records, limiting the number of records returned in a page.
5.1.3 When specifying records to be returned, they should be done from a durable identifier describing the record (as opposed to the index or position of the record in the collection)
5.1.4 Paginated responses must use the following envelope structure:

```
{
  "data": [ ... ],
  "pagination": {
  "nextCursor": "opaque-token-value"
  }
}
```

5.1.4.1 `data` contains the array of returned resources
5.1.4.2 `nextCursor` is an opaque string token used to retrieve the next page
5.1.4.3 When no further pages exist, `nextCursor` must be `null`

5.1.4.4 The internal format of `nextCursor` is left to the implementer

6. Request & Response Formatting

6.1 JSON Format

6.1.1 All request and response bodies use JSON

6.1.2 Content-Type header: `application/json`

6.2 Date and Time

6.2.1 DateTime in ISO-8601 format

6.2.2 All date and time are in UTC

6.3 Descriptive Payloads

6.3.1 Example response:

```
{
      "deviceId": "abc-123",
      "deviceState": "ONLINE",
      "hardwareIdentity": {
      "serialNumber": "1232123456"
},
      "softwareIdentity": {
      "firmwareVersion": "4.5.0",
      "model": "Microphone"
},
      "capabilities": [
      "firmware-version",
      "audio-mute"
      ]
}
```

6.4 Extensibility

6.4.1 Client may ignore data returned that doesn't fit the spec (allows for future additions)

7. Validation & Error Handling

7.1 Validation

7.1.1 All incoming requests must be validated by the API.
7.1.2 Requests containing invalid or malformed data must be rejected.
7.1.3 Validation failures must return clear, actionable error information.

7.2 Standard Error Structure

7.2.1 Refer to RFC9457 (which obsoletes RFC7807)
7.2.2 Error responses include a consistent JSON structure.
7.2.3 Error payloads should include:
- 7.2.3.1 A machine-readable error code
- 7.2.3.2 A human-readable message
- 7.2.3.3 Optional details describing the validation or processing failure

7.2.4 Example error response:

```
{
      "type": "",
      "title": "Requested device(s) could not be
      found.",
      "errorId": 123,
      "detail": "Device information is not available",
      "instance": "",
      "traceId": "df41558a-01e8-4db1-bdfe-034de7b32c5e"
}
```

8. Documentation

8.1.1 The API must be documented using the OpenAPI initiative (formerly Swagger, now OAS) guidelines.
8.1.2 Documentation should include:
- 8.1.2.1 Endpoint definitions
- 8.1.2.2 Request and response schemas
- 8.1.2.3 Example requests and responses

8.1.3 Documentation should be suitable for both human readers and automated tooling.

9. Authentication & Security

9.1 Authentication

9.1.1 OAuth 2.0 is used for authorization and access delegation

9.1.2 Access is granted via OAuth tokens

9.1.3 Token-based access enables cross-cloud integration scenarios

9.1.4 Authentication of the resource owner or end-user is outside the scope of this specification and left to each implementation.

9.1.5 For cloud-to-cloud integration scenarios, implementations must support the Client Credentials Grant (RFC 6749 §4.4). Other grant types (e.g., Authorization Code, Device Flow) may be supported to accommodate additional use cases.

9.1.6 Access tokens must be transmitted using the Authorization request header with the Bearer scheme, as defined in RFC 6750. Example: `Authorization: Bearer <token>`

9.2 Authorization

9.2.1 Role-Based Access Control (RBAC) should be supported where applicable

9.2.2 Authorization rules may differ by capability or operation

9.3 Transport Security

9.3.1 All endpoints must use HTTPS

9.3.2 TLS 1.2+ is required for data in transit

9.3.3 Sensitive data should be encrypted at rest where applicable

10. Discovery

10.1 Discovery

10.1.1 The API should support discoverability of:

10.1.1.1 Available resources

10.1.1.2 Supported capabilities

10.2 Registration (Initial Scope)

10.2.1 Initial implementation focuses on OAuth-based access only

10.2.2 Registration and claiming of:

10.2.2.1 Users

10.2.2.2 Devices are considered future iterations and out of initial scope

11. Resources & Endpoints

11.1 Devices

11.1.1 List Devices

```
GET /v1/devices
```

11.1.1.1 Returns a paginated list of devices accessible to the caller

11.1.1.2 Supports filtering, sorting, and pagination via query parameters specified in Section 5.

11.1.1.3 Each device must minimally implement the following field as described in Section 12.

```
{
      "capabilities": [
      "firmware-version",
      "audio-mute"
      ]
}
```

11.1.2 Get Device

```
GET /v1/devices/{deviceId}
```

11.1.2.1 Returns the details of a single device accessible to the caller

11.2 Capabilities

11.2.1 Get Device Capabilities

```
GET /v1/devices/{deviceId}/capabilities
```

11.2.1.1 Returns the set of capabilities supported by the specified device

11.2.1.2 Capabilities are enumerated using a standard list where possible

11.2.1.3 Each capability must provide a mapping to its corresponding APIs listed within the Open API specification.

11.2.1.4 One recommended approach shown below demonstrates how to use the optional `operationId` field as a self-documenting means to in a human-readable and

machine-readable way understand what interactions are possible given a device's list of capabilities.

```
[
      {
            "name": "audio-mute",
            "operationIds": [
            "deviceAudioMute",
            "updateDeviceAudioMute",
            "subscribeDeviceAudioMute",
            "unsubscribeDeviceAudioMute"
            ]
      },
      {
            "name": "reboot",
            "operationIds": [
            "rebootDevice"
            ]
      }
      ...
]
```

12. Capabilities Model

12.1 Capability Enumeration

12.1.1 Each device exposes a defined list of capabilities

12.1.1.1 Capabilities allow one to write code that is feature-focused instead of device-focused. When capabilities are used to check for device features, this meets the Open-Closed Principle of SOLID when new devices are supported by the API, or features are added to existing devices.

12.1.1.2 This works on the same principle as interfaces in object-oriented programming; the interaction with the capability is considered interchangeable when...

12.1.1.2.1 Two or more devices implement the same capability

12.1.1.2.2 Two or more firmware versions of the same model of device implement the same capability

12.1.2 A future OpenAV effort will define a standardized capability registry to identify common definitions

12.1.2.1 Examples could include, but are not limited to: Audio Mute, Reboot, 802.1x Certificate Configuration

12.2 Constraints

12.2.1 Capabilities may share common semantics across devices but differ in constraints, such as:

12.2.1.1 Value ranges (e.g., volume, gain, brightness)

12.2.1.2 Allowed formats or character sets (e.g., regex constraints on names)

13. Bulk Operations & Batching

13.1.1 The API should support:

13.1.1.1 Bulk endpoints

13.1.1.2 Batch operations

13.1.2 Batching aligns with OData-style JSON conventions

13.1.3 This enables efficient updates across multiple devices or capabilities

14. Events & Subscriptions

14.1 Event-Driven Model

14.1.1 The API favors event-driven communication Polling should be avoided where possible

14.2 Subscription Mechanisms

14.2.1 Possible mechanisms include:

14.2.1.1 Webhooks

14.2.1.2 WebSockets

14.2.1.3 Event callbacks

14.2.2 Details of event schemas and lifecycle management are part of a dedicated event-driven section to be expanded

15. Future Topics (Out of Scope for initial revision)

15.1 The following topics are explicitly out of scope for the initial release but are expected to be addressed in future iterations:

15.1.1 Device claiming, provisioning, and onboarding

15.1.2 User and organization registration

15.1.3 Agent-to-agent (A2A) integration

15.1.4 Model Context Protocol (MCP) and AI agent interoperability

16. Related Specifications

The following OpenAV Cloud specifications are related to this document:

Specification	Version	Relationship
AV Device Taxonomy Guidelines	1.1	Defines the device categories and taxonomy referenced in Section 12 (Capabilities Model)
AV Device Minimum Functionality Guidelines	1.1	Defines the minimum device functionality that implementations of this API are expected to expose
AV Device Security Guidelines	1.1	Defines the security requirements that complement Section 9 (Authentication & Security) of this specification