Amazon Signage Remote Management API

Version	Release Date	Description
V1.0.0	2026.01.20	Initial release
V1.0.1	2026.03.09	Update API description.
V1.1.0	2026.04.30	Add V1.1 APIs (OTA, Rotation, CMS Restart, Enhanced Screenshot, Recording)

1. Overview

This document provides CMS partners with an introduction to Amazon Signage Remote Management API and is intended to assist you with

- understanding the overall architecture and design philosophy
- preparing your technical infrastructure for integration
- beginning your planning integration roadmap

2. Executive Summary

2.1 What is Amazon Signage Remote Management API?

The Remote Management API enables certified Content Management System (CMS) applications to directly control and monitor Signage Stick devices through on-device communication. This enables CMS providers to offer enhanced device management capabilities while maintaining security and user privacy.

2.2 Key Benefits

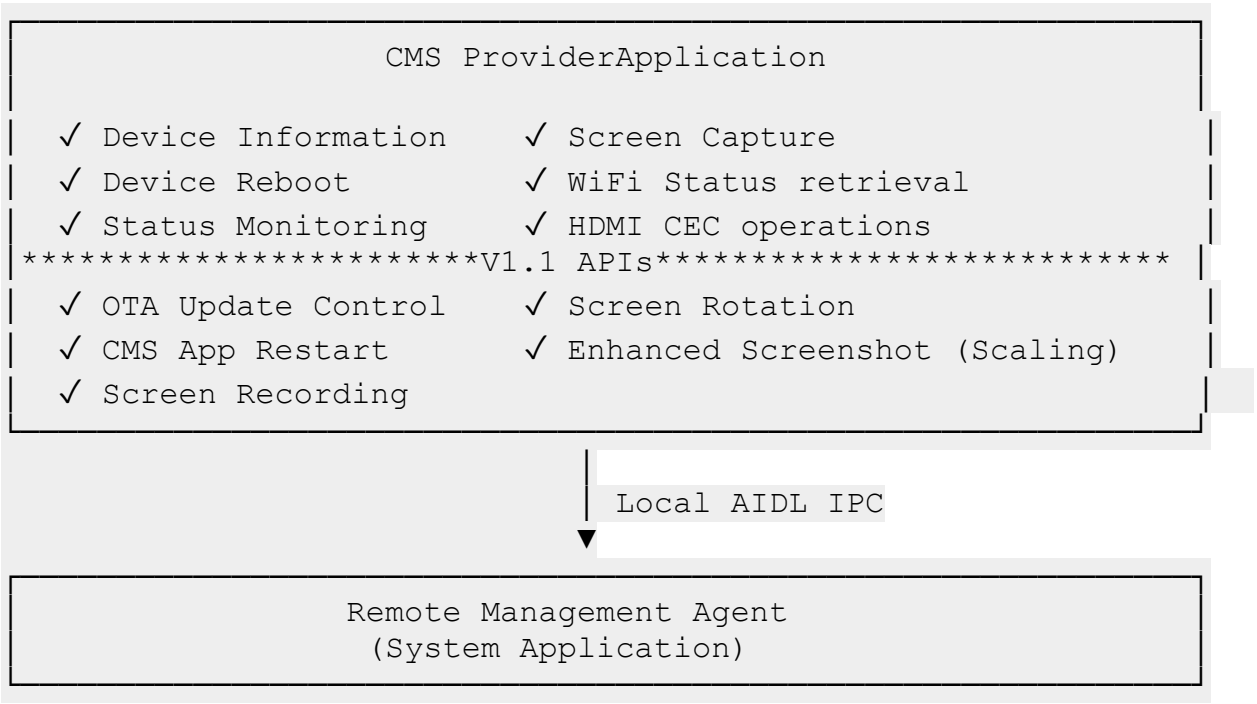
For CMS providers:

- Direct device access with minimal latency
- Customizable user experience within CMS applications
- Simplified authentication, without the need for loud credentials
- Real-time device operations and feedback

For End Users:

- Single application for content and device management
- Faster response times for device operations
- Reduced network dependency
- Improved reliability

2.3 Core Capabilities



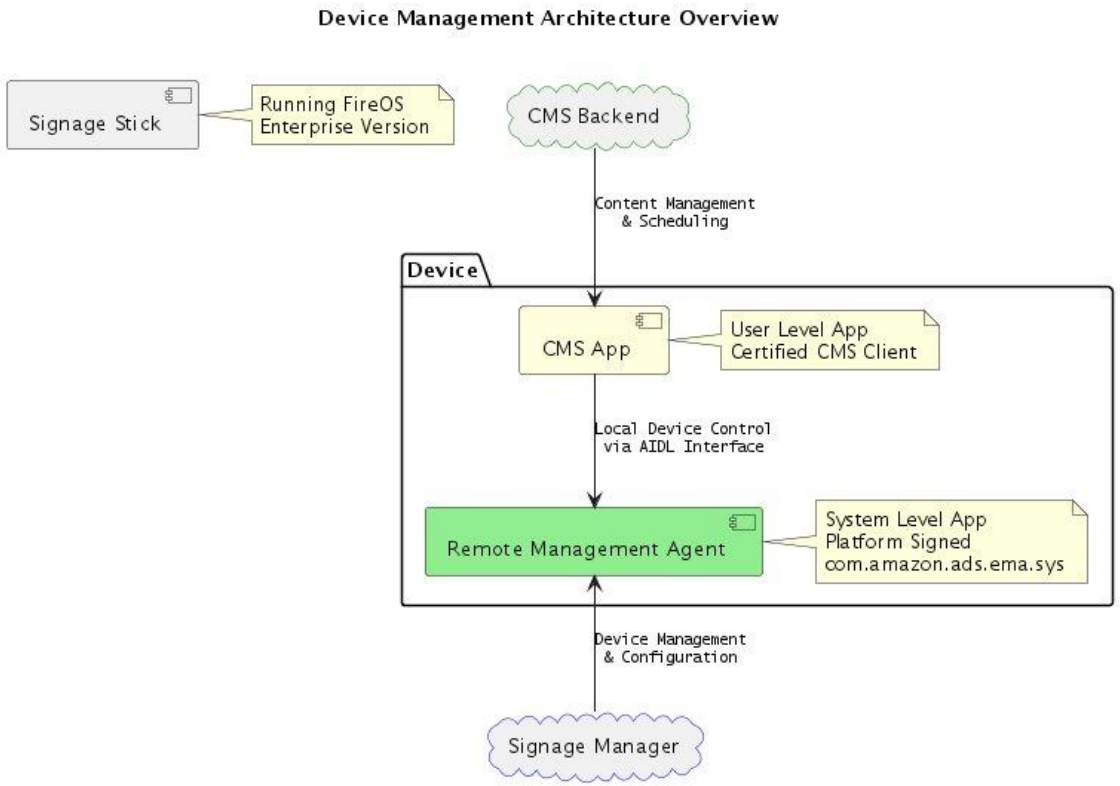
2.4 Who Should Read This Document?

- Product Managers: Sections 2, 3, 6, 8
- Technical Architects: Sections 4, 5, 7
- Developers: All sections, especially 4, 5, 6, 7

3. Background and Context

3.1 Ecosystem

The Amazon Device Solution (ADS) Signage Stick ecosystem consists of three main components:



3.2 The Solution

Device-Level Integration using Android IPC (Inter-Process Communication) mechanism:

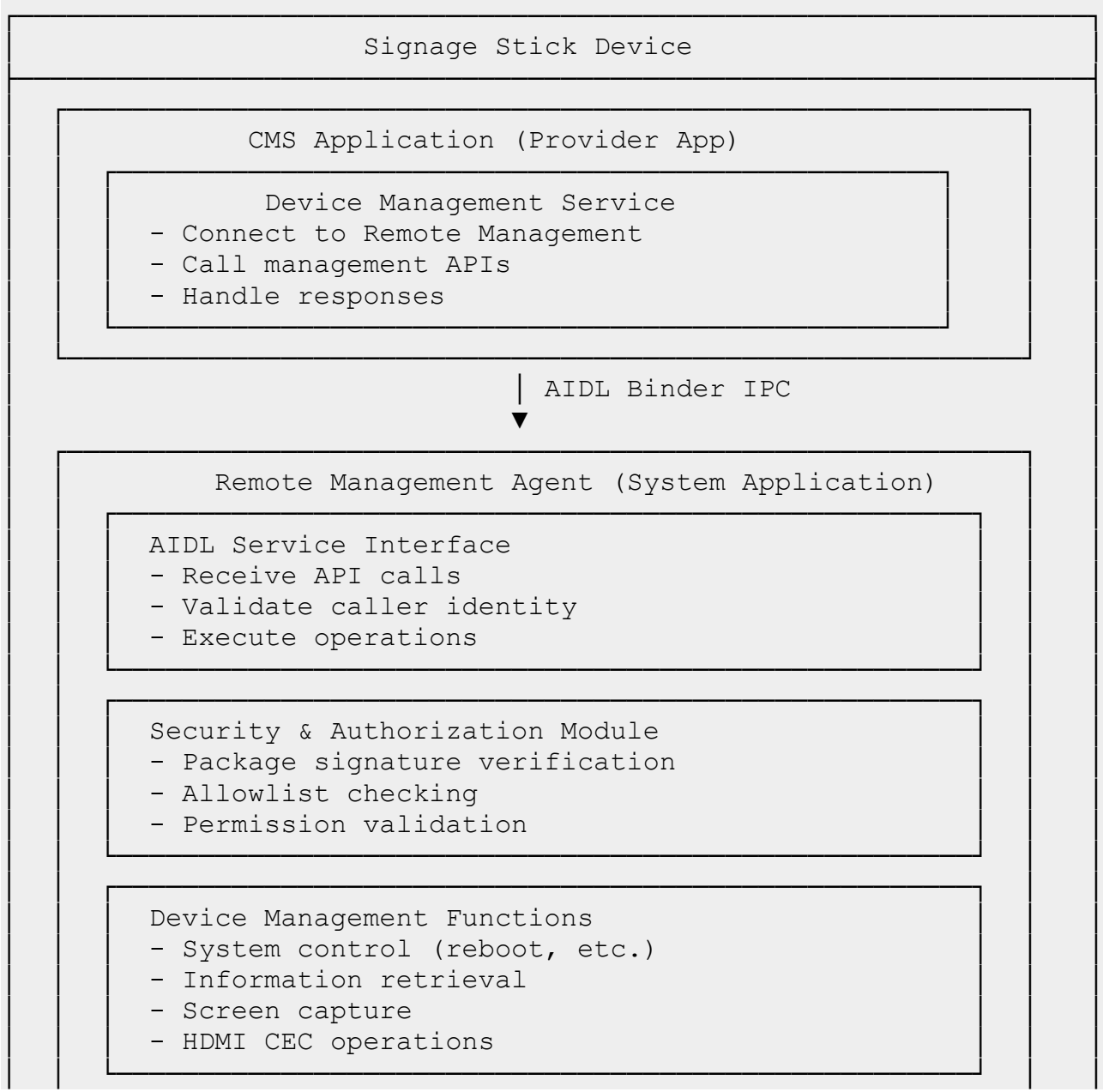
`CMS App → Remote Management → Device Action (<100ms)`

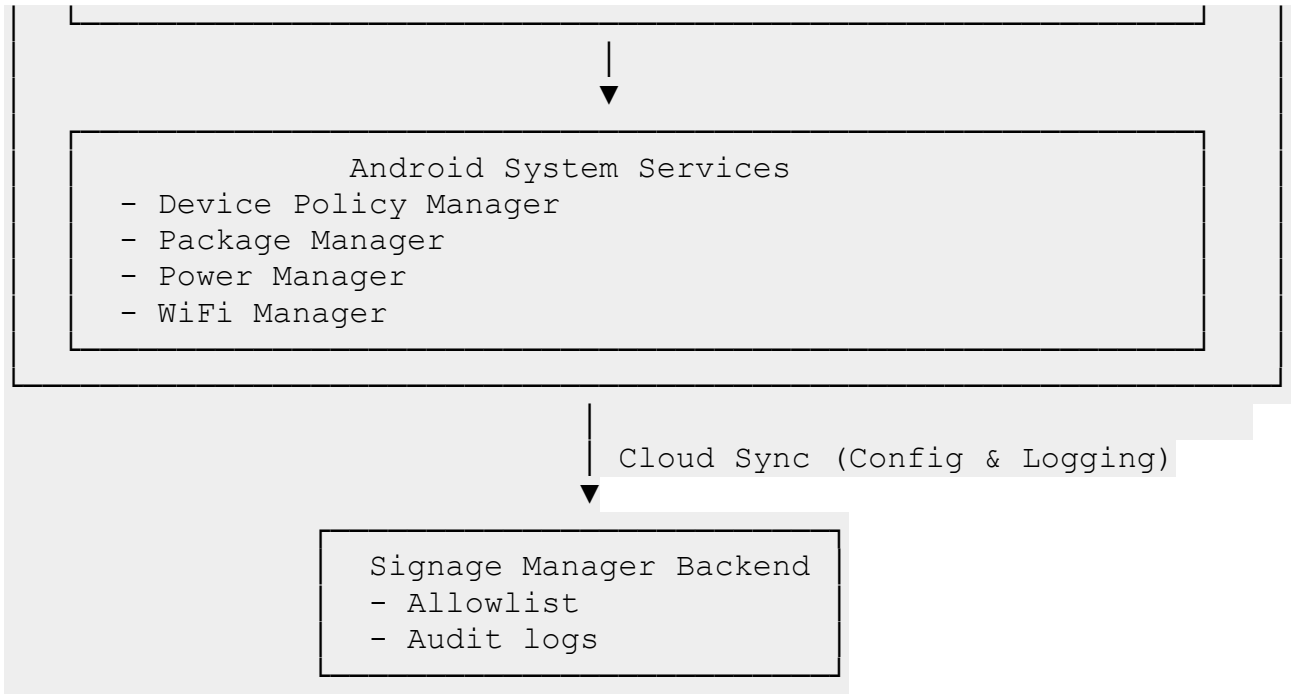
This approach provides:

- 10-50x faster response times compared to Cloud-to-Cloud based solution
- Offline capability for most operations
- Simplified integration with standard Android patterns
- Better security through local validation

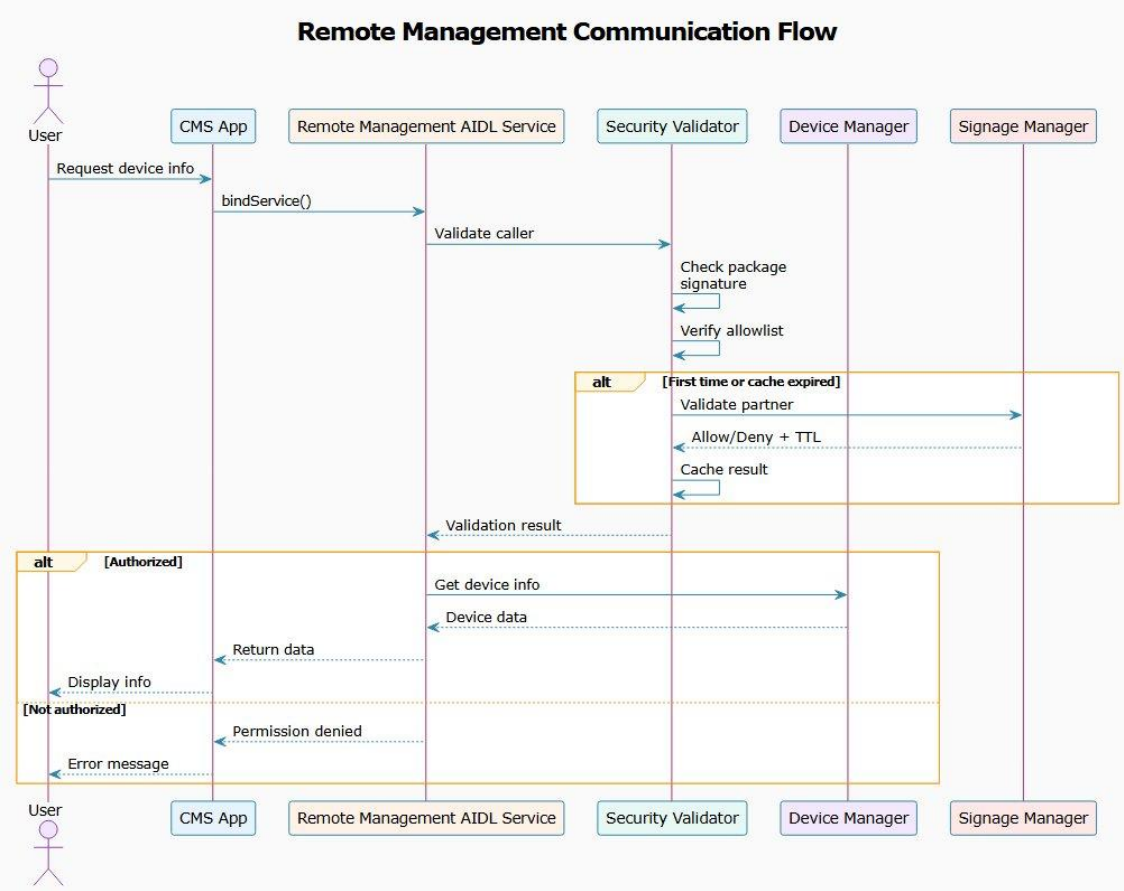
4. Solution Architecture

4.1 High-Level Architecture




Android System Services
- Device Policy Manager
- Package Manager
- Power Manager
- WiFi Manager
Cloud Sync (Config & Logging)
Signage Manager Backend
- Allowlist
- Audit logs

4.2 Communication Flow



4.3 Technology Stack

Android IPC Mechanism:

- AIDL (Android Interface Definition Language): Standard Android IPC
- Binder: Low-level Android communication infrastructure
- Service Binding: Android service connection pattern

Key Characteristics:

- Cross-process communication
- Type-safe method calls
- Built-in permission system
- Automatic lifecycle management

5 API Capabilities Overview

5.1 Available Operations

The Remote Management API will provide the following capabilities:

5.1.1 Device Information

```
Device Identification
• Device Serial Number (DSN)
• Model name and hardware info
• OS version and build number
```

```
System Status
• CPU usage
• Memory usage (total, free, used, ratio)
• Storage usage (total, free, used, ratio)
```

```
Hardware Monitoring
• CPU temperature (Celsius)
• GPU temperature (Celsius)
• Thermal status (NORMAL/WARNING/CRITICAL)
```

```
Network Status
• WiFi connection state
• Signal strength (dBm and level 0-4)
• Link speed (Mbps)
• Connected SSID
• IP address and MAC address
```

```
HDMI & Display Status
• HDMI connection state
• EDID data (raw hex string)
• Monitor info (parsed from EDID)
• Display manufacturer and model
• Resolution and refresh rate
```

Note: All data structures align with REST API format. Values are in bytes (not MB), field names match REST API conventions. See Appendix B for complete data models.

5.1.2 Device Control

```
System Operations
• Reboot device (with optional delay 0-300 seconds)
• Factory reset device (with external storage option)
```

```
HDMI CEC Operations
• Get HDMI connection and CEC status
• Refresh CEC status (query power and audio)
• Toggle display power (on/off)
• Get display information from EDID
```

```
Screen Capture
• Capture current screen
• Configurable format (PNG, JPEG, WEBP)
• Quality control (1-100, JPEG only)
• Returns image data with metadata
```

```
Maintenance Operations
• Clear application caches (specific apps)
```

```
**********V1.1 APIs**********
```

```
OTA Update Control
•  getUpdateStatus
•  checkCmsUpdate
•  checkOsUpdate
```

```
Screen Rotation
•  getScreenRotation
•  setScreenRotation
```

```
CMS App Restart
• restartCms
```

```
Enhanced Screenshot (Scaling)
• captureScreenScaled
• getScreenshotChunk
• clearScreenshot
```

```
Screen Recording
• startScreenRecord
• getScreenRecordStatus
• getScreenRecordChunk
• clearScreenRecord
```

Important HDMI CEC Notes:

- CEC commands are asynchronous - responses arrive after 100-500ms
- Always call `refreshHdmiStatus()` before reading power/audio state
- Not all displays support all CEC commands - check `canControl` flag

See Appendix B for complete CEC API details and implementation examples

5.2 Sample API Calls (Conceptual)

Note: These are simplified conceptual examples. For complete implementation including error handling, async execution, and full code examples, please refer to Appendix B: AIDL Interface Definitions.

Getting Device Information:

```
// Connect to Remote Management service
EmaDeviceManager deviceManager = new EmaDeviceManager(context);
deviceManager.connect();

// Get device info
DeviceInfo info = deviceManager.getDeviceInfo();
if (info.code == 200) {
    String dsn = info.data.dsn;
    String version = info.data.versionName;
    long uptimeMs = info.data.elapsedMs;
```

```
}

// Get system resources
SystemResources resources = deviceManager.getSystemResources();
if (resources.code == 200) {
    long memoryFreeMB = resources.data.memory.free / (1024 * 1024);
    float memoryUsagePercent = resources.data.memory.ratio;
    float cpuTemp = resources.data.temperature.cpuCelsius;
}
```

Controlling HDMI Display:

```
// Get current HDMI status
HdmiStatus status = deviceManager.getHdmiStatus();
if (status.code == 200 && status.data.connected) {
    boolean displayOn = status.data.power.displayOn;
}

// Refresh CEC status (async operation)
deviceManager.refreshHdmiStatus();
Thread.sleep(500);  // Wait for CEC response
HdmiStatus updatedStatus = deviceManager.getHdmiStatus();

// Toggle display power
ApiResponse response = deviceManager.toggleDisplayPower();
if (response.code == 200) {
    // Command sent successfully
}
```

Capturing Screen:

```
ScreenshotResult result = deviceManager.captureScreen("PNG", 100);
if (result.code == 200) {
    byte[] imageData = result.data.imageData;
    int width = result.data.widthPx;
    int height = result.data.heightPx;

    // Convert to Bitmap
    Bitmap bitmap = BitmapFactory.decodeByteArray(
        imageData, 0, imageData.length
```

```
    );
}
```

Rebooting Device:

```
// Immediate reboot
ApiResponse response = deviceManager.rebootDevice(0);

// Delayed reboot (30 seconds)
ApiResponse response = deviceManager.rebootDevice(30);

if (response.code == 200) {
    // Reboot scheduled successfully
    Map<String, Object> data = response.data;
    String executeAt = (String) data.get("executeAt");
}
```

For complete usage examples including:

- Full async implementation with AsyncTask/Coroutines
- Comprehensive error handling
- Service lifecycle management
- UI integration examples
- Best practices and patterns

See Appendix B.4 - Service Connection Implementation

5.3 Data Formats (Preliminary)

All API responses follow a consistent RESTful structure aligned with the existing REST API:
Standard Response Structure:

```
{
  "code": 200,
  "message": "OK",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": { }
}
```

Key Format Specifications:

	Field	Format	Example	Notes
1	code	Integer	200, 401, 500	HTTP-style status codes

2	message	String	"OK", "Unauthorized"	Human-readable status
3	timestamp	ISO 8601	"2025-10-11T23:01:00.000Z"	With timezone info
4	Memory/Storage	Bytes	1743544320	Not MB - matches REST API
5	Temperature	Float	45.5	Celsius
6	Uptime	Long	4887494	Milliseconds
7	Signal Strength	Integer	-45	dBm
8	Volume	Integer	0-100	Percentage

Sample Response Examples:
For complete JSON response examples for all APIs, including:

- Device information responses
- System resources responses
- HDMI status responses (connected and not connected)
- Control operation responses
- Error responses

See Appendix B.3 - JSON Response Examples

5.4 Error Handling

All API errors follow RESTful conventions with consistent error structure:

```
{
  "code": 401,
  "message": "Unauthorized",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "error": "Signature verification failed",
    "packageName": "com.unknown.app",
    "reason": "Package not in allowlist"
  }
}
```

Common Error Codes:

	Code	Name	Description	Recommended Action
1	200	OK	Request succeeded	Continue processing
2	400	Bad Request	Invalid parameters	Check parameter values and types
3	401	Unauthorized	Signature verification failed	Verify app signature and allowlist entry
4	403	Forbidden	Permission denied	Check device enrollment and permissions

5	404	Not Found	Resource or feature not found	Check if feature is supported
6	429	Too Many Requests	Rate limit exceeded	Implement backoff and retry logic
7	500	Internal Server Error	Operation failed	Log error and retry after delay
8	501	Not Implemented	Feature not implemented	Check feature support, graceful fallback
9	503	Service Unavailable	Remote Management service unavailable	Retry after delay, check service status
10	504	Gateway Timeout	Operation timed out	Increase timeout or retry

Error Handling Best Practices:

```
try {
    ApiResponse response = deviceManager.toggleDisplayPower();

    switch (response.code) {
        case 200:
            // Success
            handleSuccess(response);
            break;
        case 401:
            // Unauthorized - signature issue
            handleUnauthorized(response);
            break;
        case 501:
            // Feature not available (e.g., CEC not supported)
            handleFeatureNotAvailable(response);
            break;
        case 503:
            // Service unavailable - retry
            scheduleRetry();
            break;
        default:
            handleUnknownError(response);
    }
} catch (RemoteException e) {
    // IPC communication error
    handleCommunicationError(e);
}
```

5.5 Detailed API Specifications

This section provides an overview of the API structure and conventions. For complete AIDL interface definitions, data models, and implementation examples, please refer to Appendix B: AIDL Interface Definitions.

5.5.1 API Design Principles

All APIs follow RESTful conventions with consistent response structure:
Standard Response Format:

```
{
  "code": 200,                                  // HTTP-style status code
  "message": "OK",                              // Human-readable message
  "timestamp": "2025-10-11T23:01:00.000Z",  // ISO 8601 with timezone
  "data": {}                                    // API-specific response data
}
```

Status Code Categories:

- 2xx (Success): 200 (OK), 201 (Created), 204 (No Content)
- 4xx (Client Error): 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), 429 (Too Many Requests)
- 5xx (Server Error): 500 (Internal Server Error), 501 (Not Implemented), 503 (Service Unavailable), 504 (Gateway Timeout)

5.5.2 Data Structure Alignment

The Local Partner API data structures are fully aligned with REST API to ensure consistency:
Memory and Storage:

- Values in bytes (not MB) - matches REST API format
- Field names: `total`, `free`, `used`, `ratio` - matches REST API naming
- Example from REST API:

```
"memory": {
  "total": 1743544320,
  "free": 560029696,
  "used": 1183514624,
  "ratio": 67.879814
}
```

Device Information:

- Includes all REST API fields: `dsn`, `emaVersion`, `versionName`, `versionCode`, `kioskApp`, `timezone`, `monitorInfo`, `edid`, `elapsedMs`

- EDID and monitor info available in both device info and HDMI status
- Uptime tracked as `elapsedMs` (milliseconds since boot)

Timestamp Format:

- ISO 8601 with timezone: `"2025-10-11T23:01:00.000Z"`
- Human-friendly and machine-parseable
- Consistent across all API responses

5.5.3 API Categories

The Local Partner API provides five main categories of operations:

1. Device Information APIs
   - `getDeviceInfo()` - DSN, version, build info, uptime
   - `getSystemResources()` - Memory, storage, CPU, temperature
   - `getNetworkStatus()` - WiFi connection, signal strength, IP address
   - `getHdmiStatus()` - HDMI connection, CEC status, display info
2. Device Control APIs
   - `rebootDevice()` - Reboot with optional delay
   - `factoryResetDevice()` - Factory reset (with warning)

3. HDMI CEC Control APIs
   - `toggleDisplayPower()` - Toggle display on/off
   - `refreshHdmiStatus()` - Query current CEC status
4. Screen Capture API
   - `captureScreen()` - Capture with format and quality options
   - `captureScreenScaled()` - Capture current screen with optional resolution scaling
5. Maintenance APIs
   - `clearCache()` - System and app cache clearing

5.5.4 HDMI CEC Implementation Notes

The HDMI CEC APIs are based on the Fire OS HDMI utility implementation (see `HdmiUtils.java`):

CEC Command Mapping:

- `toggleDisplayPower()` → Sends CEC `USER_CONTROL_PRESSED` with `POWER` code
- `refreshHdmiStatus()` → Sends `GIVE_DEVICE_POWER_STATUS` and `GIVE_AUDIO_STATUS` queries

Important CEC Considerations:

- CEC response is asynchronous - status updates arrive ~100-500ms after query
- Not all displays support all CEC commands
- Always check `canControl` flag in status responses

- Use `refreshHdmiStatus()` before reading power/audio state for latest values

5.5.5 Rate Limiting

To ensure system stability, the following rate limits apply:

	API Category	Rate Limit	Notes
1	Device Information	10 requests/minute	Cache results when possible
2	Control APIs (reboot, reset)	1 request/5 minutes	Safety limitation
3	HDMI CEC APIs	5 requests/minute	Avoid CEC bus congestion
4	Cache Clear	1 request/10 minutes	Resource-intensive operation
5	Screen Capture	5 requests/minute	CPU/memory intensive
6	Exceeding rate limits will result in HTTP 429 (Too Many Requests) error code.		

5.5.6 Best Practices

Caching:

- Cache device information (DSN, version) - rarely changes
- Cache system resources for 10-30 seconds - reduces API calls
- Don't cache HDMI status - always refresh before reading

Error Handling:

- Always check `code` field in responses
- Handle all documented error codes gracefully
- Implement retry logic with exponential backoff for 5xx errors
- Log errors for debugging but don't spam logs

Async Operations:

- Use background threads (AsyncTask, Executor, Coroutines) for all API calls
- Never call APIs on main/UI thread
- Show loading indicators during operations
- Provide user feedback on success/failure

HDMI CEC:

- Call `refreshHdmiStatus()` before reading power/audio state
- Wait 500ms after refresh before calling `getHdmiStatus()`
- Check `canControl` flag before attempting control operations
- Handle CEC unavailable (code 501) gracefully

Resource Management:

- Bind to service when needed, unbind when done
- Don't maintain permanent connection unless actively using
- Clean up in `onDestroy()` or similar lifecycle methods

5.5.7 Complete API Reference

For complete details including:

- Full AIDL interface definitions
- All data model parcelables
- JSON response examples
- Service connection implementation
- Complete usage examples
- Project structure and build configuration

Please see Appendix B: AIDL Interface Definitions.

6. Implementation Status

6.1 Implementation Categories

The current API implementation falls into these categories:

	Status	Description	Example APIs	Release phase
1	✅ Fully Implemented	Returns real, live data from device	`getDeviceInfo()`, `getNetworkStatus()`	2026,Jan
2	🚧 Simulated Data	Returns sample data for testing only	`getSystemResources() - temp + cpu`	Will be replaced with real data in API v1 .2

Note: Methods marked with status "Simulated Data" are for future use. They can be used for implementation testing purposes but won't return live data until API v 1.2

6.2 Detailed Implementation Status

6.2.1 Device Information APIs

getDeviceInfo() - ☑ Fully Implemented

- All fields return real device data
- DSN, version info, timezone, uptime are accurate
- EDID data reflects actual HDMI connection state

getSystemResources() - 🚧 Simulated Data

```
{
  "memory": {
    "total": 1743544320,      // ☑ Real value
    "free": 560029696,        // ☑ Real value
    "used": 1183514624,       // ☑ Real value
    "ratio": 67.879814        // ☑ Real value
```

```
  },
  "storage": {
    "total": 12520574976,     // ☑ Real value
    "free": 12058910720,      // ☑ Real value
    "used": 461664256,        // ☑ Real value
    "ratio": 3.6872451        // ☑ Real value
  },
  "temperature": {
    "cpuCelsius": 45.5,       // 🚧 Simulated (random 40-50°C)
    "gpuCelsius": 48.2,       // 🚧 Simulated (random 45-55°C)
    "thermalState": "NORMAL"  // 🚧 Always returns "NORMAL"
  },
  "cpu": {
    "usagePercent": 15.6      // 🚧 Simulated (random 10-30%)
  }
}
```

getNetworkStatus() - ☑ Fully Implemented

- WiFi connection state, signal strength, IP address are real
- All network information reflects actual device status

6.2.2 Device Control APIs

rebootDevice() - ☑ Fully Implemented

- Device will actually reboot after specified delay
- All functionality works as documented

factoryResetDevice() - ☑ Fully Implemented

- Returns success response

clearCache() - ☑ Fully Implemented

- Returns realistic freed space values and perform actual cache clearing based on input options
- Operation duration and app count are simulated

6.2.3 HDMI CEC APIs

Overview: HDMI CEC functionality is implemented via Fire OS's `AmazonHdmiServiceManager`, providing power control and display information retrieval. However, due to CEC protocol compatibility differences across devices, some features have limitations on specific TV brands.
Power Control APIs

	API	Fire TV	Samsung TV	Notes
1	`setDisplayPower(true)`	✅ Fully Working	✅ Fully Working	Uses multiple CEC commands for compatibility
2	`setDisplayPower(false)`	✅ Fully Working	✅ Fully Working	Standard STANDBY command
3	`toggleDisplayPower()`	✅ Fully Working	✅ Fully Working	Toggles between on/off states

getHdmiStatus() - ☑ Fully Implemented (some fields have limitations)

- `connected`: ✅ Real-time HDMI connection detection
- `display.manufacturer/model/resolution`: ✅ Parsed from EDID, accurate and reliable
- `power.displayOn`: ✅ Power status query available
- `power.canControl`: ✅ All CEC devices support power control

CMS Integration Recommendations

1. Power Control: Safe to use, best compatibility across all devices

6.3 Testing Recommendations

6.3.1 What You Can Test Now

☑ API Integration
- Service binding and connection
- Authentication and authorization
- Response format and error handling
- Rate limiting behavior

☑ Real Device Data
- Device identification (DSN, version)
- Network connectivity status
- HDMI connection detection
- Memory and storage information

☑ Core Device Control
- Device reboot functionality
- Permission system behavior

Added from API V1.1 :

6.4.1 OTA Update Control

Description

CMS applications can query update status, trigger CMS APK updates, and device OS updates via Partner API. Two modes are supported:

- `forceApply=false`: Triggers a check only, follows normal OTA flow (subject to time window restrictions)
- `forceApply=true`: Creates a force update transaction, immediately downloads and installs, bypassing time window

API List

	Method	Description	Return
1	`getUpdateStatus()`	Query CMS and OS update status	UpdateStatus
2	`checkCmsUpdate(boolean forceApply)`	Check/force CMS update	ApiResponse
3	`checkOsUpdate(boolean forceApply)`	Check/force OS update (device reboots)	ApiResponse

Response Codes

	Code	Meaning
1	200	Operation initiated
2	409	A force update transaction is already in progress
3	503	Device not ready (OOBE incomplete, CMS not configured, etc.)

Update Status Values

`none` → `pending` → `downloading` → `downloaded` → `installing` → `completed` / `failed`

Impact Scope

- EMA Plus: `DeviceManagementService`, `OtaUpdateHandler`, `OtaUpdateMgr`, `DataStore`, `ForceUpdateTransaction`
- Demo APP: New OTA Update tab
- Does not affect normal OTA scheduled task flow

Test Points

1. getUpdateStatus — Returns `none` when no update available; returns correct version and status when update exists
2. checkCmsUpdate(false) — Returns 200 after triggering check, does not install immediately
3. checkCmsUpdate(true) — Immediately downloads and installs when update available; creates pending transaction and triggers check when no update info exists
4. checkOsUpdate(true) — Downloads, installs, then device reboots
5. Duplicate calls — Returns 409 when a transaction already exists
6. Device not ready — Returns 503 when OOBE is incomplete

7. Download failure — Transaction marked as failed, getUpdateStatus shows failure reason
8. Transaction auto-cleanup — Cleared 5 seconds after completion, subsequent OTA resumes normal logic
9. Progress query — getUpdateStatus returns downloadProgress 0-100 during download

6.4.2. Screen Rotation

Description

CMS applications can query and set the device screen rotation. Setting rotation automatically disables auto-rotation.

API List

	Method	Description
1	`getScreenRotation()`	Get current rotation, response data contains `rotation` and `degrees`
2	`setScreenRotation(String rotation)`	Set rotation direction

Rotation Values

	Value	Degrees	Description
1	`landscape`	0°	Landscape (default)
2	`portrait`	90°	Portrait
3	`reverse_landscape`	180°	Reverse landscape
4	`reverse_portrait`	270°	Reverse portrait

Impact Scope

- EMA Plus: `ScreenRotationHandler`, implemented via `Settings.System.USER_ROTATION`
- Setting rotation also disables `accelerometer_rotation` (auto-rotate)

Test Points

1. getScreenRotation — Returns current rotation, default `landscape`
2. setScreenRotation — Test all four directions, verify screen actually rotates
3. Invalid value — Invalid string returns 400
4. Persistence — Rotation persists after device reboot
5. Auto-rotation — Auto-rotation is disabled after setting

6.4.3. CMS App Restart

Description
CMS applications can call this API to restart themselves. EMA Plus force-stops the caller process, waits 500ms, then relaunches it.

API

	Method	Description
1	`restartCms()`	Restart the calling CMS application

The caller's packageName is automatically determined from Binder caller identity — no parameters needed.

Impact Scope
- EMA Plus: Implemented directly in `DeviceManagementService`, calls `AdminUtils.killApp()` + `AdminUtils.startApp()`

Test Points
1. Normal restart — CMS app is killed and relaunched after calling
2. Unauthorized caller — Unauthorized app returns 401
3. Post-restart state — Partner API connection must be re-established after restart

6.4.4. Enhanced Screenshot (captureScreenScaled)

Description

Adds `captureScreenScaled(format, quality, maxWidth)` on top of the existing `captureScreen(format, quality)`, supporting proportional scaling.

API

	Method	Parameters	Description
1	`captureScreenScaled(String format, int quality, int maxWidth)`	format: PNG/JPEG/WEBP, quality: 1-100, maxWidth: max width in pixels	Proportionally scaled screenshot

- `maxWidth=0` or negative: Full resolution (same as captureScreen)
- `maxWidth=960`: 1920x1080 device outputs 960x540

Binder Chunked Transfer
Android Binder transactions have a ~1MB size limit. When screenshot data exceeds 900KB:
1. `captureScreen` / `captureScreenScaled` returns `ScreenshotResult` with `data.chunked=true`, `data.imageData=null`, `data.totalSize` set to total size
2. Client calls `getScreenshotChunk(offset)` to retrieve data in 512KB chunks

3. After download completes, calls `clearScreenshot()` to free server-side cache

When screenshot is ≤900KB, data is returned directly in `data.imageData` with `chunked=false` — no chunking needed.

New APIs

	Method	Description
1	`getScreenshotChunk(long offset)`	Get screenshot data chunk at offset (≤512KB per chunk)
2	`clearScreenshot()`	Clear server-side cached screenshot data

Impact Scope

- EMA Plus: `ScreenCaptureHandler` (new caching and chunking logic), `ScreenshotData` (new `chunked`/`totalSize` fields), `ScreenshotChunk` (new model)
- Demo APP: `DisplayFragment` automatically handles chunked download, new Quality input field
- AIDL: New `ScreenshotChunk` model and 2 methods

Test Points

1. Small image inline — JPEG quality=50 maxWidth=480, returns `chunked=false`, imageData directly usable
2. Large image chunked — PNG full resolution (1080p), returns `chunked=true`, retrieve complete data via chunk API
3. Chunk integrity — Reassembled chunk data matches original image, decodes to valid Bitmap
4. clearScreenshot — After calling, getScreenshotChunk returns 404
5. Quality parameter — Different JPEG/WEBP quality values affect file size; PNG ignores quality
6. MaxWidth scaling — maxWidth=960 outputs resolution 960xN (proportional)
7. Format validation — Invalid format returns 400
8. Demo APP — Format/Quality/MaxWidth all configurable in UI

6.4.5. Screen Recording

Description

CMS applications can record the device screen (1-20 seconds). After recording completes, the video file is retrieved via chunked transfer.

API List

	Method	Description
1	`startScreenRecord(ScreenRecordConfig config)`	Start recording (non-blocking)
2	`getScreenRecordStatus()`	Query recording status
3	`getScreenRecordChunk(long offset)`	Get video data chunk at offset (≤512KB per chunk)
4	`clearScreenRecord()`	Delete recorded video file

ScreenRecordConfig Parameters

	Field	Type	Default	Description
1	durationSeconds	int	5	Recording duration 1-20 seconds
2	resolution	String	"native"	"720p" / "1080p" / "native"
3	bitrateMbps	int	4	Video bitrate 1-8 Mbps

Recording States

`idle` → `recording` → `completed` / `failed`

Usage Flow

```
startScreenRecord(config) → wait → getScreenRecordStatus() until completed
→ getScreenRecordChunk(0) → getScreenRecordChunk(512KB) → ... until isLastChunk
→ clearScreenRecord()
```

Impact Scope

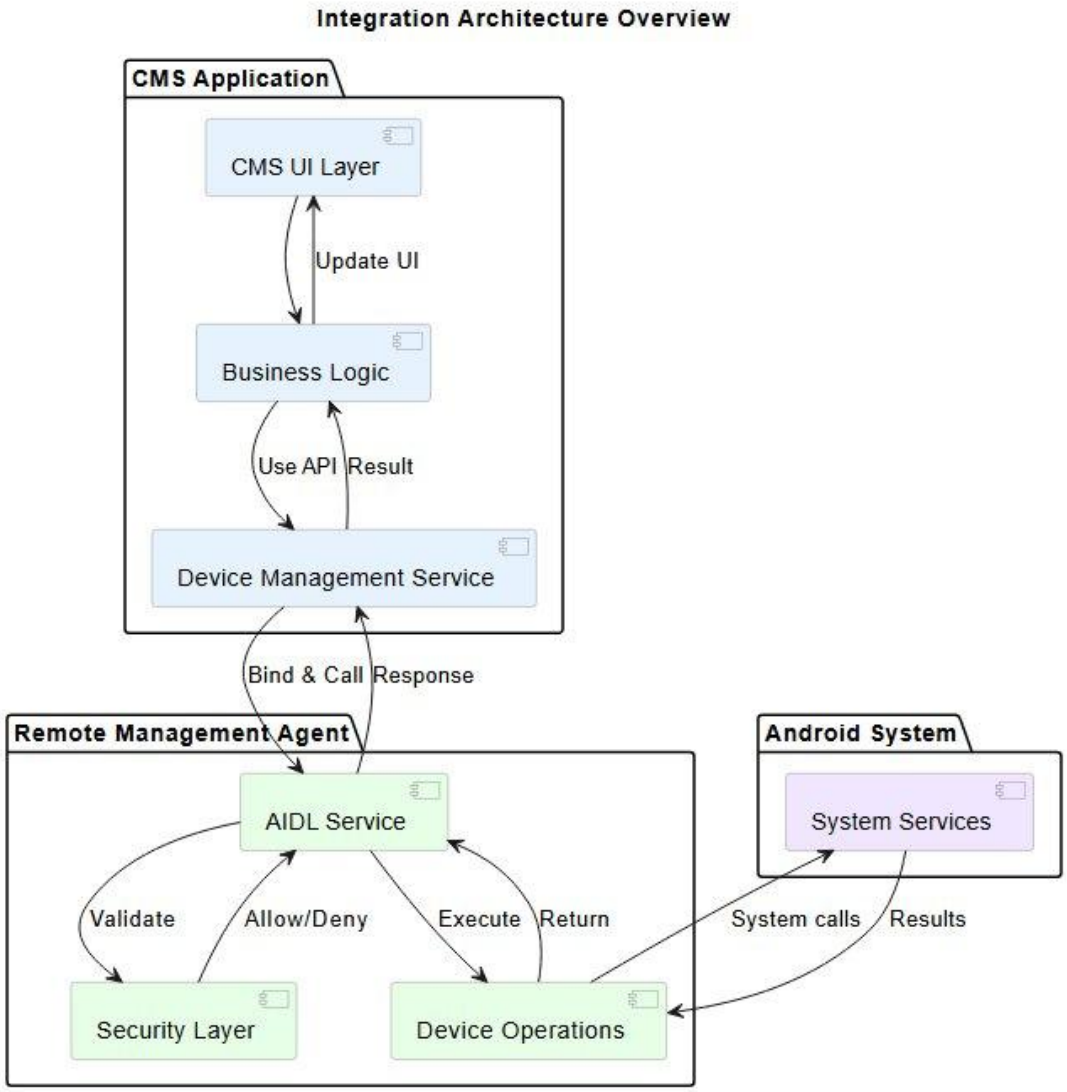
- EMA Plus: `ScreenRecordHandler`, uses Android `MediaRecorder` + `MediaProjection`
- Demo APP: Screen Recording section in Display tab

Test Points

1. Basic recording — Default parameters, 5-second recording, status transitions from recording → completed
2. Custom parameters — Various duration/resolution/bitrate combinations
3. Duplicate recording — Calling during active recording returns 409
4. Chunk download — Video file fully downloaded via chunk API
5. Cleanup — clearScreenRecord deletes the file
6. Demo APP — Duration/Resolution configurable in UI, auto-downloads and plays after recording

7. Integration Approach

7.1 Integration Architecture



7.2 API Distribution

ADS will provide:

Remote Management API:

```
Remote Management API
├── AIDL interface definitions
├── Helper classes for service binding
├── Data models for requests/responses
├── Error handling utilities
└── Sample code and documentation
```

Integration Steps (refer to B.8 Reference demo project for the fully workable implementation) :

1. Add dependency to your `build.gradle`
2. Declare required permissions in `AndroidManifest.xml`
3. Dynamically request DEVICE_MANAGEMENT permission
4. Initialize Local API in your application
5. Bind to Remote Management service
6. Make API calls as needed

CMS need to provide:

Please provide the following information to the ADS team:

```
Application Information:
├── Package name (e.g., com.yourcompany.cms)
├── Application signing certificate SHA-256 digest (see below)
│   ├── Debug build signature (for testing)
│   └── Release build signature (for production) ReSigned by FireTV Appstore
├── Certificate Authority (if applicable)
├── Target Android API level
```

Debug vs Release Signatures:

Your app will have different signatures for debug and release builds:

1. For Testing Phase: Provide debug build signature
2. For Production Phase: Provide release build signature
3. Both: If you want to test with production builds

Signature Format:

- ✅ Lowercase hexadecimal: `c8a2e9bccf597c2fb6dc66bee293fc13f2fc47ec77bc6b2b0d52c11f51192ab8`
- ❌ Uppercase with colons: `C8:A2:E9:BC:CF:59:7C:2F:...`
- ❌ Base64 encoded: `yKLpvM9ZfC+23Ga...`

Appendix A: Glossary

	Term	Definition
1	ADS	Amazon Device Solution - The team developing enterprise Fire OS products
2	AIDL	Android Interface Definition Language - Android's IPC mechanism
3	APK	Android Package - Application installation file
4	Certificate Digest	SHA-256 hash of the signing certificate
5	CMS	Content Management System - Software for managing digital signage content
6	DSN	Device Serial Number - Unique identifier for each device
7	Remote Management	Endpoint Management Agent Plus - System app for device management
8	IPC	Inter-Process Communication - Mechanism for apps to communicate
9	SHA-256	Secure Hash Algorithm 256-bit - Cryptographic hash function
10	Signature Scheme	APK signing method (v1, v2, or v3)
11	Signage Manager	Cloud management platform for Signage Stick devices
12	TTL	Time To Live - Cache expiration duration

Appendix B: AIDL Interface Definitions

This section provides the complete AIDL interface definitions for the Remote Management API.

B.1 Main Service Interface

File: `IDeviceManagement.aidl`

```
package com.amazon.ads.ema.api;
import com.amazon.ads.ema.api.model.DeviceInfo;
import com.amazon.ads.ema.api.model.SystemResources;
import com.amazon.ads.ema.api.model.NetworkStatus;
import com.amazon.ads.ema.api.model.HdmiStatus;
import com.amazon.ads.ema.api.model.ScreenshotResult;
import com.amazon.ads.ema.api.model.ScreenshotChunk;
import com.amazon.ads.ema.api.model.ScreenRecordConfig;
import com.amazon.ads.ema.api.model.ScreenRecordStatus;
import com.amazon.ads.ema.api.model.ScreenRecordChunk;
import com.amazon.ads.ema.api.model.CacheClearOptions;
import com.amazon.ads.ema.api.model.CacheClearResult;
import com.amazon.ads.ema.api.model.WifiConfig;
import com.amazon.ads.ema.api.model.ApiResponse;
import com.amazon.ads.ema.api.model.TimezoneListResult;
import com.amazon.ads.ema.api.model.UpdateStatus;
```

```
/**
 * EMA Local Partner API
 * Main interface for device management operations.
 *
 * All responses follow RESTful convention:
 * - code: HTTP-style status code (200 = success, 4xx = client error, 5xx = server error)
 * - message: Human-readable status message
 * - timestamp: ISO 8601 format with timezone (e.g., "2025-10-11T23:01:00.000Z")
 * - data: Response data as Map or typed object
 *
 * Method ordering is significant for AIDL transaction codes.
 * Must match the server AIDL exactly.
 *
 * @version 1.1
 * @since 2026-Q1
 */
interface IDeviceManagement {
    // ============================================================
    // [v1.0] Device Information APIs
    // ============================================================
    /**
     * Get device serial number (DSN) and basic information
     * @return DeviceInfo with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    DeviceInfo getDeviceInfo();
    /**
     * Get system resource information (CPU, memory, storage, temperature)
     * @return SystemResources with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    SystemResources getSystemResources();
    /**
     * Get network status and WiFi information
     * @return NetworkStatus with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    NetworkStatus getNetworkStatus();
    /**
```

```
     * Get HDMI connection and CEC status
     * Note: Power and audio states are from last CEC query.
     * Call refreshHdmiStatus() to update with current values.
     *
     * @return HdmiStatus with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    HdmiStatus getHdmiStatus();
    /**
     * Refresh HDMI power and audio status by sending CEC queries
     * This triggers CEC GIVE_DEVICE_POWER_STATUS and GIVE_AUDIO_STATUS commands.
     * Status will be updated asynchronously when device responds.
     * Call getHdmiStatus() after ~500ms to get updated values.
     *
     * @return ApiResponse with code 200 if queries sent successfully
     */
    ApiResponse refreshHdmiStatus();
    // ============================================================
    // [v1.0] Device Control APIs
    // ============================================================
    /**
     * Reboot the device
     * @param delaySeconds Delay before reboot (0 for immediate, max 300)
     * @return ApiResponse with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     * @throws IllegalArgumentException if delaySeconds invalid (code 400)
     */
    ApiResponse rebootDevice(int delaySeconds);
    /**
     * Factory reset the device
     * WARNING: This will erase all user data and cannot be undone
     *
     * @param wipeExternalStorage Whether to wipe external storage
     * @return ApiResponse with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    ApiResponse factoryResetDevice(boolean wipeExternalStorage);
    /**
     * Wake up the device screen
```

```
     * @return ApiResponse with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    ApiResponse wakeScreen();
    // ============================================================
    // [v1.0] HDMI CEC Control APIs
    // ============================================================
    /**
     * Toggle display power via HDMI CEC
     * Sends CEC POWER command (toggles between ON and STANDBY)
     *
     * @return ApiResponse with code 200 if command sent
     *         code 501 if CEC not available
     *         code 503 if HDMI not connected
     */
    ApiResponse toggleDisplayPower();
    /**
     * Set display power state via HDMI CEC
     * Note: Not all displays support explicit ON/OFF commands.
     * Consider using toggleDisplayPower() for better compatibility.
     *
     * @param powerOn true to turn on, false to turn off
     * @return ApiResponse with code 200 if command sent
     *         code 501 if CEC not available or command not supported
     */
    ApiResponse setDisplayPower(boolean powerOn);
    /**
     * Adjust volume via HDMI CEC
     * Sends a single volume up/down command
     *
     * @param volumeUp true for volume up, false for volume down
     * @return ApiResponse with code 200 if command sent
     */
    ApiResponse adjustVolume(boolean volumeUp);
    /**
     * Toggle mute state via HDMI CEC
     * Sends CEC MUTE command (toggles between muted and unmuted)
     *
     * @return ApiResponse with code 200 if command sent
```

```
     */
    ApiResponse toggleMute();
    /**
     * Set volume level via HDMI CEC
     * Note: Not all displays support absolute volume control via CEC.
     * This may require multiple volume up/down commands.
     *
     * @param level Volume level (0-100)
     * @return ApiResponse with code 200 if attempting to set
     *         code 400 if level out of range
     *         code 501 if not supported
     */
    ApiResponse setVolumeLevel(int level);
    // ============================================================
    // [v1.0] Screen Capture API
    // ============================================================
    /**
     * Capture current screen
     * @param format Image format: "PNG", "JPEG", or "WEBP"
     * @param quality Quality (1-100, only affects JPEG and WEBP)
     * @return ScreenshotResult with code 200 and image data on success
     *         code 400 if parameters invalid
     */
    ScreenshotResult captureScreen(String format, int quality);
    // ============================================================
    // [Feature] Network Configuration API
    // ============================================================
    /**
     * Configure WiFi network
     * NOTE: Placeholder only; not officially released yet. Do not use in production.
     *
     * @param config WiFi configuration
     * @return ApiResponse with code 200 on success
     *         code 400 if config invalid
     */
    ApiResponse configureWifi(in WifiConfig config);
    // ============================================================
    // [v1.0] Maintenance APIs
    // ============================================================
```

```
    /**
     * Clear system and app caches
     * @param options Cache clear options
     * @return CacheClearResult with code 200 and freed space info
     */
    CacheClearResult clearCache(in CacheClearOptions options);
    // ============================================================
    // [v1.0] Utility APIs
    // ============================================================
    /**
     * Get API version
     * @return Version string (e.g., "1.1.0")
     */
    String getApiVersion();
    /**
     * Check if specific feature is supported on this device
     * Features: "HDMI_CEC", "SCREENSHOT", "WIFI_CONFIG", "CACHE_CLEAR",
     *           "OTA_UPDATE", "SCREEN_ROTATION", "SCREEN_RECORD"
     *
     * @param feature Feature name
     * @return true if supported, false otherwise
     */
    boolean isFeatureSupported(String feature);
    // ============================================================
    // [v1.1] OTA Update APIs
    // ============================================================
    /**
     * Get current CMS and OS update status, including any active force update transactions
     * @return UpdateStatus with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    UpdateStatus getUpdateStatus();
    /**
     * Check for CMS APK update
     * Normal check respects time-window rules configured by the MDM.
     * Force apply starts an async transaction: check -> download -> install immediately.
     * Poll getUpdateStatus() to track progress.
     *
     * @param forceApply false for a normal check, true to force immediate apply
```

```
     * @return ApiResponse with code 200 if request accepted
     *         code 409 if another update transaction is already in progress
     */
    ApiResponse checkCmsUpdate(boolean forceApply);
    /**
     * Check for device OS update
     * Normal check respects time-window rules configured by the MDM.
     * Force apply starts an async transaction: check -> download -> install immediately.
     * Device will reboot after installation when forceApply is true.
     * Poll getUpdateStatus() to track progress.
     *
     * @param forceApply false for a normal check, true to force immediate apply
     * @return ApiResponse with code 200 if request accepted
     *         code 409 if another update transaction is already in progress
     */
    ApiResponse checkOsUpdate(boolean forceApply);
    // ============================================================
    // [v1.1] Screen Rotation API
    // ============================================================
    /**
     * Get current screen rotation
     * @return ApiResponse with code 200; data is one of
     *         "landscape", "portrait", "reverse_landscape", "reverse_portrait"
     */
    ApiResponse getScreenRotation();
    /**
     * Set screen rotation
     * Disables auto-rotation.
     *
     * @param rotation One of "landscape", "portrait", "reverse_landscape", "reverse_portrait"
     * @return ApiResponse with code 200 on success
     *         code 400 if rotation value is invalid
     */
    ApiResponse setScreenRotation(String rotation);
    // ============================================================
    // [v1.1] Enhanced Screen Capture API
    // ============================================================
    /**
     * Capture current screen with optional resolution scaling
```

```
     * Enhanced version of captureScreen() with maxWidth for proportional downscaling.
     *
     * @param format Image format: "PNG", "JPEG", or "WEBP"
     * @param quality Quality (1-100, only affects JPEG and WEBP)
     * @param maxWidth Maximum width in pixels for proportional scaling.
     *                 Use 0 or negative to capture at full device resolution.
     * @return ScreenshotResult with code 200 and image data on success
     *         code 400 if parameters invalid
     */
    ScreenshotResult captureScreenScaled(String format, int quality, int maxWidth);
    // ============================================================
    // [v1.1] CMS App Control API
    // ============================================================
    /**
     * Reboot (force-stop and relaunch) the calling CMS application
     * @return ApiResponse with code 200 on success
     * @throws SecurityException if caller is not authorized (code 401)
     */
    ApiResponse restartCms();
    // ============================================================
    // [Future] Screen Recording API
    // ============================================================
    /**
     * Start screen recording
     * Non-blocking call. Poll getScreenRecordStatus() to track progress.
     *
     * @param config Screen recording configuration (resolution, bitrate, duration)
     * @return ScreenRecordStatus with code 200 if recording started
     *         code 409 if a recording is already in progress
     */
    ScreenRecordStatus startScreenRecord(in ScreenRecordConfig config);
    /**
     * Get current screen recording status and progress
     * @return ScreenRecordStatus with code 200 on success
     */
    ScreenRecordStatus getScreenRecordStatus();
    /**
     * Get a chunk of recorded video data
     * Call repeatedly with increasing offset until isLastChunk is true.
```

```
     * Each chunk is up to 512KB.
     *
     * @param offset Byte offset into the recorded file
     * @return ScreenRecordChunk with code 200 and chunk data on success
     */
    ScreenRecordChunk getScreenRecordChunk(long offset);
    /**
     * Delete the recorded video file to free storage
     * @return ApiResponse with code 200 on success
     */
    ApiResponse clearScreenRecord();
    // ============================================================
    // [Future] Timezone Configuration API
    // ============================================================
    /**
     * Get list of all available timezones with display information
     * NOTE: Placeholder only; not officially released yet. Do not use in production.
     *
     * @return TimezoneListResult with code 200 and the list on success
     */
    TimezoneListResult getTimezoneList();
    /**
     * Set device timezone (disables automatic timezone detection)
     * NOTE: Placeholder only; not officially released yet. Do not use in production.
     *
     * @param timezoneId IANA timezone ID (e.g., "America/Los_Angeles")
     * @return ApiResponse with code 200 on success
     *         code 400 if timezoneId is invalid
     */
    ApiResponse setTimezone(String timezoneId);
    // ============================================================
    // [v1.1] Screenshot Chunk Transfer API
    // ============================================================
    /**
     * Get screenshot image data chunk
     * Used when captureScreen/captureScreenScaled returns chunked=true.
     * Call repeatedly with increasing offset until isLastChunk is true.
     * Each chunk is up to 512KB.
     *
```

```
     * @param offset Byte offset into the captured screenshot
     * @return ScreenshotChunk with code 200 and chunk data on success
     */
    ScreenshotChunk getScreenshotChunk(long offset);
    /**
     * Clear cached screenshot data to free memory
     * @return ApiResponse with code 200 on success
     */
    ApiResponse clearScreenshot();
}
```

B.2 Data Model AIDL Files

File: `ApiResponse.aidl`

```
package com.amazon.ads.ema.api.model;
/**
 * Standard API response following RESTful conventions
 * Matches common REST API structure: code, message, data
 */
parcelable ApiResponse {
    int code;  // Status code (200 = success, 4xx = client error, 5xx = server error)
    String message;          // Human-readable message
    String timestamp;        // ISO 8601 format: "2025-10-11T23:01:00.000Z"
    java.util.Map data;      // Response data as key-value pairs
}
```

File: `DeviceInfo.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable DeviceInfo {
    int code;                   // Status code
    String message;             // Status message
    String timestamp;           // ISO 8601: "2025-10-11T23:01:00.000Z"
    DeviceData data;            // Device data
}
parcelable DeviceData {
    String dsn;                 // Device Serial Number
    String kioskApp;            // Current kiosk/CMS app package name
    String timezone;            // Timezone (e.g., "Asia/Singapore-GMT+08:00")
```

```
    String monitorInfo;         // Monitor info (e.g., "DELL SP2418H-1920x1080")
    String edid;                // Raw EDID hex string (all zeros if no HDMI)
    long elapsedMs;             // Device uptime in milliseconds
}
```

File: `SystemResources.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable SystemResources {
    int code;                   // Status code
    String message;             // Status message
    String timestamp;           // ISO 8601: "2025-10-11T23:01:00.000Z"
    ResourceData data;          // Resource data
}
parcelable ResourceData {
    MemoryInfo memory;
    StorageInfo storage;
    TemperatureInfo temperature;
    CpuInfo cpu;
    long elapsedMs;             // Device uptime in milliseconds
}
parcelable MemoryInfo {
    long total;                 // Total memory in bytes
    long free;                  // Free memory in bytes
    long used;                  // Used memory in bytes
    float ratio;                // Usage ratio (0-100)
}
parcelable StorageInfo {
    long total;                 // Total internal storage in bytes
    long free;                  // Free storage in bytes
    long used;                  // Used storage in bytes
    float ratio;                // Usage ratio (0-100)
}
parcelable TemperatureInfo {
    float cpuCelsius;           // CPU temperature in Celsius
    float gpuCelsius;           // GPU temperature in Celsius
    String thermalState;        // "NORMAL", "WARNING", "CRITICAL"
}
parcelable CpuInfo {
```

```
    float usagePercent;           // Overall CPU usage (0-100)
}
```

File: `NetworkStatus.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable NetworkStatus {
    int code;               // Status code
    String message;         // Status message
    String timestamp;       // ISO 8601: "2025-10-11T23:01:00.000Z"
    NetworkData data;       // Network data
}
parcelable NetworkData {
    WifiInfo wifi;
    EthernetInfo ethernet;
}
parcelable WifiInfo {
    boolean connected;
    String ssid;
    int signalStrengthDbm;// Signal strength in dBm (e.g., -45)
    int signalLevel;      // Signal level 0-4 (4 = excellent)
    int linkSpeedMbps;    // Link speed in Mbps
    int frequency;        // Frequency in MHz (2400-2500 or 5000-6000)
    String ipAddress;
    String macAddress;
}
parcelable EthernetInfo {
    boolean connected;
    String ipAddress;
    String macAddress;
}
```

File: `HdmiStatus.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable HdmiStatus {
    int code;                // Status code
    String message;          // Status message
    String timestamp;        // ISO 8601: "2025-10-11T23:01:00.000Z"
```

```
    HdmiData data;            // HDMI data
}
parcelable HdmiData {
    boolean connected;        // HDMI cable connected (based on EDID)
    String edid;              // Raw EDID hex string (all zeros if not connected)
    String monitorInfo;       // Parsed monitor info (e.g., "DELL SP2418H-1920x1080")
    boolean cecEnabled;       // CEC is enabled
    boolean activeSource;     // Device is active source
    DisplayInfo display;      // Parsed display info (null if not connected)
    PowerStatus power;        // Power status (from CEC query)
    AudioStatus audio;        // Audio status (from CEC query)
}
parcelable DisplayInfo {
    String manufacturer;      // Display manufacturer (from EDID)
    String model;             // Display model (from EDID)
    String resolution;        // e.g., "1920x1080"
    int refreshRate;          // e.g., 60 (Hz)
}
parcelable PowerStatus {
    boolean displayOn;    // Display power state (from CEC query)
    String state;         // "ON", "STANDBY", "TRANSITIONING_ON", "TRANSITIONING_OFF", "UNKNOWN"
    boolean canControl;   // Can control via CEC
    long lastUpdatedMs;   // Last query timestamp (epoch millis)
}
parcelable AudioStatus {
    int volume;               // Volume level 0-100 (from CEC query)
    boolean muted;            // Mute state (from CEC query)
    boolean canControl;       // Can control via CEC
    long lastUpdatedMs;       // Last query timestamp (epoch millis)
}
```

File: `ScreenshotResult.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable ScreenshotResult {
    int code;                 // Status code
    String message;           // Status message
    String timestamp;         // ISO 8601: "2025-10-11T23:01:00.000Z"
    ScreenshotData data;      // Screenshot data (null if failed)
```

```
}
parcelable ScreenshotData {
    byte[] imageData;        // Raw image bytes
    String format;           // "PNG" or "JPEG" or "WEBP"
    int widthPx;             // Image width in pixels
    int heightPx;            // Image height in pixels
    long fileSizeBytes;      // Image size in bytes
}
```

File: `WifiConfig.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable WifiConfig {
    String ssid;
    String password;
    String securityType;     // "OPEN", "WEP", "WPA", "WPA2_PSK", "WPA3"
    boolean hidden;
    int priority;
    StaticIpConfig staticIp; // TBD, null for DHCP
}
parcelable StaticIpConfig {
    String ipAddress;
    String gateway;
    String dns1;
    String dns2;
    String subnetMask;
}
```

File: `CacheClearOptions.aidl`

```
package com.amazon.ads.ema.api.model;
parcelable CacheClearOptions {
    boolean systemCache;     // Clear system cache
    boolean appCache;        // Clear all app caches
    String[] packageNames;   // Specific packages (null for all)
}
```

File: `CacheClearResult.aidl`

```
package com.amazon.ads.ema.api.model;

parcelable CacheClearResult {
    int code;                    // Status code
    String message;              // Status message
    String timestamp;            // ISO 8601: "2025-10-11T23:01:00.000Z"
    CacheClearData data;         // Cache clear data
}
parcelable CacheClearData {
    long systemCacheFreedMB;     // System cache freed in MB
    long appCacheFreedMB;        // App cache freed in MB
    int appsProcessed;           // Number of apps processed
    long totalFreedMB;           // Total freed space in MB
    long durationMs;             // Operation duration in milliseconds
}
```

B.3 JSON Response Examples

Example: getDeviceInfo() Response


```
{
  "code": 200,
  "message": "OK",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "dsn": "G072JM0834641AKK",
    "emaVersion": "v1.7.3-118",
    "versionName": "2.3.0",
    "versionCode": 29,
    "kioskApp": "tv.ablesign.app",
    "timezone": "Asia/Singapore-GMT+08:00",
    "monitorInfo": "null-1920x1080",
    "edid": "00000000000000000000000000000000000000.....",
    "elapsedMs": 4887494
  }
}
```


Example: getSystemResources() Response

```
{
  "code": 200,
  "message": "OK",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "memory": {
      "total": 1743544320,
      "free": 560029696,
      "used": 1183514624,
      "ratio": 67.879814
    },
    "storage": {
      "total": 12520574976,
      "free": 12058910720,
      "used": 461664256,
      "ratio": 3.6872451
    },
    "temperature": {
      "cpuCelsius": 45.5,
      "gpuCelsius": 48.2,
      "thermalState": "NORMAL"
    },
    "cpu": {
      "usagePercent": 15.6
    },
    "elapsedMs": 4887494
  }
}
```


Example: getHdmiStatus() Response (No HDMI Connected)


```
{
  "code": 200,
  "message": "OK",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "connected": false,
    "edid": "0000000000000000....",
    "monitorInfo": "null-1920x1080",
```

```
    "cecEnabled": false,
    "activeSource": false,
    "display": null,
    "power": {
      "displayOn": false,
      "state": "UNKNOWN",
      "canControl": false,
      "lastUpdatedMs": 0
    },
    "audio": {
      "volume": 0,
      "muted": false,
      "canControl": false,
      "lastUpdatedMs": 0
    }
  }
}
```


Example: getHdmiStatus() Response (HDMI Connected)


```
{
  "code": 200,
  "message": "OK",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "connected": true,
    "edid": "00ffffffffffff0010ac00a1423235301e1b0103.....",
    "monitorInfo": "DELL SP2418H-1920x1080",
    "cecEnabled": true,
    "activeSource": true,
    "display": {
      "manufacturer": "DELL",
      "model": "SP2418H",
      "resolution": "1920x1080",
      "refreshRate": 60
    },
    "power": {
      "displayOn": true,
      "state": "ON",
```

```
      "canControl": true,
      "lastUpdatedMs": 1728684060000
    },
    "audio": {
      "volume": 15,
      "muted": false,
      "canControl": true,
      "lastUpdatedMs": 1728684060000
    }
  }
}
```

Example: toggleDisplayPower() Response

```
{
  "code": 200,
  "message": "CEC command sent successfully",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "action": "toggleDisplayPower",
    "executed": true,
    "command": "POWER_TOGGLE"
  }
}
```

Example: rebootDevice() Response

```
{
  "code": 200,
  "message": "Reboot scheduled",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "action": "reboot",
    "scheduled": true,
    "executeAt": "2025-10-11T23:01:30.000Z",
    "delaySeconds": 30
  }
}
```

Example: Error Response (Unauthorized)

```
{
  "code": 401,
  "message": "Unauthorized",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "error": "Signature verification failed",
    "packageName": "com.unknown.app",
    "reason": "Package not in allowlist"
  }
}
```

Example: Error Response (CEC Not Available)

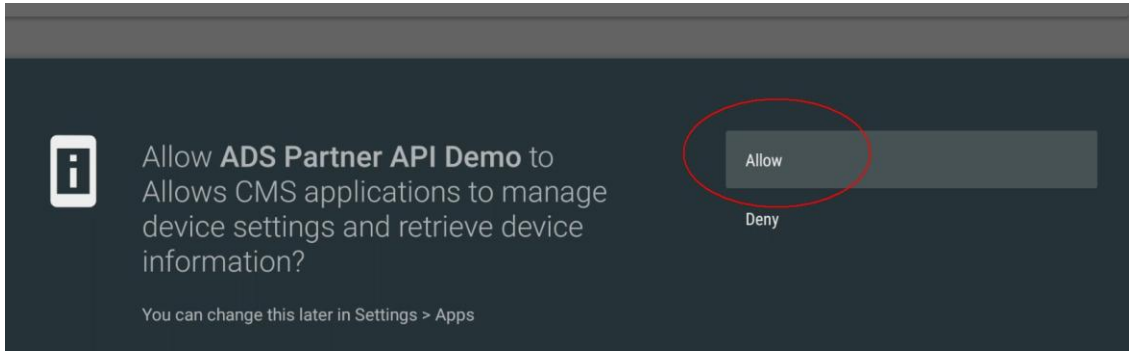
```
{
  "code": 501,
  "message": "Feature not implemented",
  "timestamp": "2025-10-11T23:01:00.000Z",
  "data": {
    "error": "HDMI CEC control is not available",
    "feature": "HDMI_CEC",
    "reason": "Display does not support CEC or HDMI not connected"
  }
}
```

B.4 Service Connection Implementation

EMA Device Manager Wrapper (refer to B.8 Reference demo project for the fully workable implementation) :

```
// Permission request code
private static final int REQUEST_DEVICE_MANAGEMENT_PERMISSION = 1001;
private static final String DEVICE_MANAGEMENT_PERMISSION
        = "com.amazon.ads.ema.permission.DEVICE_MANAGEMENT";

...

// Check and request permission before connecting to API
checkAndRequestPermission();

...

/**
```

```
* Check if DEVICE_MANAGEMENT permission is granted and request if needed
*/
private void checkAndRequestPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (ContextCompat.checkSelfPermission(this, DEVICE_MANAGEMENT_PERMISSION)
                != PackageManager.PERMISSION_GRANTED) {
            L.i("DEVICE_MANAGEMENT permission not granted, requesting...");

            // Check if we should show an explanation
            if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                    DEVICE_MANAGEMENT_PERMISSION)) {
                // Show explanation dialog before requesting permission
                showPermissionExplanationDialog();
            } else {
                // Directly request permission
                requestDeviceManagementPermission();
            }
        } else {
            L.i("DEVICE_MANAGEMENT permission already granted");
            // Permission already granted, connect to API
            connectToPartnerApi();
        }
    } else {
        // For API level < 23, permission is granted at install time
        L.i("API level < 23, no runtime permission needed");
        connectToPartnerApi();
    }
}
...
```



Note: This manually grant step is only needed during the debug/onboarding phase since the Remote Management Agent will automatically grant all the runtime permissions for the certified CMS agents.

```
package com.partner.cms;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.os.RemoteException;
import android.util.Log;

import com.amazon.ads.ema.api.IDeviceManagement;
import com.amazon.ads.ema.api.model.*;

public class EmaDeviceManager {
    private static final String TAG = "EmaDeviceManager";
    private static final String EMA_PACKAGE = "com.amazon.ads.ema.sys";
    private static final String EMA_SERVICE
            = "com.amazon.ads.ema.sys.DeviceManagementService";

    private Context context;
    private IDeviceManagement deviceService;
    private boolean bound = false;
    private ConnectionListener connectionListener;

    private final ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
```

```
        public void onServiceConnected(ComponentName name, IBinder service) {
            deviceService = IDeviceManagement.Stub.asInterface(service);
            bound = true;
            Log.i(TAG, "Connected to Remote Management service");

            if (connectionListener != null) {
                connectionListener.onConnected();
            }
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            deviceService = null;
            bound = false;
            Log.w(TAG, "Disconnected from Remote Management service");

            if (connectionListener != null) {
                connectionListener.onDisconnected();
            }
        }
    };
    public EmaDeviceManager(Context context) {
        this.context = context.getApplicationContext();
    }
    /**
     * Bind to Remote Management service
     */
    public boolean connect() {
        if (bound) {
            Log.w(TAG, "Already connected");
            return true;
        }
        Intent intent = new Intent();
        intent.setClassName(EMA_PACKAGE, EMA_SERVICE);
        try {
            bound = context.bindService(
                intent,
                serviceConnection,
                Context.BIND_AUTO_CREATE
```

```
            );
            return bound;
        } catch (SecurityException e) {
            Log.e(TAG, "Permission denied when binding to service", e);
            return false;
        }
    }
    /**
     * Disconnect from service
     */
    public void disconnect() {
        if (bound) {
            context.unbindService(serviceConnection);
            bound = false;
            deviceService = null;
        }
    }
    /**
     * Check if connected
     */
    public boolean isConnected() {
        return bound && deviceService != null;
    }
    // ============================================================
    // Device Information APIs
    // ============================================================
    public DeviceInfo getDeviceInfo() throws RemoteException {
        ensureConnected();
        return deviceService.getDeviceInfo();
    }
    public SystemResources getSystemResources() throws RemoteException {
        ensureConnected();
        return deviceService.getSystemResources();
    }
    public NetworkStatus getNetworkStatus() throws RemoteException {
        ensureConnected();
        return deviceService.getNetworkStatus();
    }
    public HdmiStatus getHdmiStatus() throws RemoteException {
```

```
        ensureConnected();
        return deviceService.getHdmiStatus();
    }
    public ApiResponse refreshHdmiStatus() throws RemoteException {
        ensureConnected();
        return deviceService.refreshHdmiStatus();
    }
    // ============================================================
    // Device Control APIs
    // ============================================================
    public ApiResponse rebootDevice(int delaySeconds) throws RemoteException {
        ensureConnected();
        return deviceService.rebootDevice(delaySeconds);
    }
    public ApiResponse factoryResetDevice(boolean wipeExternal)
            throws RemoteException {
        ensureConnected();
        return deviceService.factoryResetDevice(wipeExternal);
    }
    public ApiResponse wakeScreen() throws RemoteException {
        ensureConnected();
        return deviceService.wakeScreen();
    }
    // ============================================================
    // HDMI CEC Control APIs
    // ============================================================
    public ApiResponse toggleDisplayPower() throws RemoteException {
        ensureConnected();
        return deviceService.toggleDisplayPower();
    }
    // ============================================================
    // Other APIs
    // ============================================================
    public ScreenshotResult captureScreen(String format, int quality)
            throws RemoteException {
        ensureConnected();
        return deviceService.captureScreen(format, quality);
    }
    public CacheClearResult clearCache(CacheClearOptions options)
```

```
            throws RemoteException {
        ensureConnected();
        return deviceService.clearCache(options);
    }

    public String getApiVersion() throws RemoteException {
        ensureConnected();
        return deviceService.getApiVersion();
    }

    public boolean isFeatureSupported(String feature) throws RemoteException {
        ensureConnected();
        return deviceService.isFeatureSupported(feature);
    }

    private void ensureConnected() {
        if (!isConnected()) {
            throw new IllegalStateException(
                    "Not connected to Remote Management service");
        }
    }
    // Connection listener interface
    public interface ConnectionListener {
        void onConnected();
        void onDisconnected();
    }
    public void setConnectionListener(ConnectionListener listener) {
        this.connectionListener = listener;
    }
}
```

B.5 AndroidManifest.xml Requirements

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.partner.cms">
    <!-- Standard Android permissions -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

```
    <!-- !!!!! Required permission to access Remote Management API !!!!! -->
    <uses-permission android:name="com.amazon.ads.ema.permission.DEVICE_MANAGEMENT" />
    <application
        android:name=".CMSApplication"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher">
        <activity android:name=".CMSActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

B.6 Build Configuration

build.gradle (app level):

```
android {
    compileSdkVersion 30
    defaultConfig {
        applicationId "com.partner.cms"
        minSdkVersion 28
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"
    }
    sourceSets {
        main {
            // Include AIDL files
            aidl.srcDirs = ['src/main/aidl']
        }
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
                    'proguard-rules.pro'
```

```
        }
    }
}
dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
}
```

proguard-rules.pro:

Data structures related to the remote management API **must not** be obfuscated by ProGuard and need to be excluded.

```
# Keep AIDL interfaces
-keep interface com.amazon.ads.ema.api.** { *; }
-keep class com.amazon.ads.ema.api.model.** { *; }

# Keep Parcelables
-keepclassmembers class * implements android.os.Parcelable {
    public static final android.os.Parcelable$Creator CREATOR;
}
```

B.7 Project Structure

```
your-cms-app/
├── src/
│   └── main/
│       ├── aidl/
│       │   └── com/
│       │       └── amazon/
│       │           └── ads/
│       │               └── ema/
│       │                   └── api/
│       │                       ├── IDeviceManagement.aidl
│       │                       └── model/
│       │                           ├── ApiResponse.aidl
│       │                           ├── DeviceInfo.aidl
│       │                           ├── SystemResources.aidl
│       │                           ├── NetworkStatus.aidl
│       │                           ├── HdmiStatus.aidl
│       │                           ├── ScreenshotResult.aidl
```

```
│       │                                   ├── WifiConfig.aidl
│       │                                   ├── CacheClearOptions.aidl
│       │                                   └── (... other aidl files)
│       ├── java/
│       │   └── com/
│       │       └── amazon/
│       │           └── ads/
│       │               └── ema/
│       │                   └── api/
│       │                       └── model/
│       │                           ├── ApiResponse.java
│       │                           ├── DeviceInfo.java
│       │                           ├── SystemResources.java
│       │                           ├── NetworkStatus.java
│       │                           ├── HdmiStatus.java
│       │                           ├── ScreenshotResult.java
│       │                           ├── WifiConfig.java
│       │                           ├── CacheClearOptions.java
│       │                           └── (... other java model files)
│       │           └── partner/
│       │               └── cms/
│       │                   ├── EmaDeviceManager.java
│       │                   ├── CMSActivity.java
│       │                   └── (... other app code)
│       ├── res/
│       │   ├── layout/
│       │   │   └── activity_cms.xml
│       │   └── values/
│       │       └── strings.xml
│       └── AndroidManifest.xml
├── build.gradle
└── proguard-rules.pro
```

B.8 Reference demo project

https://file.signage.amazon.com/src/ads-cli-android-partner-api-demo-src.tar
This project is a fully compilable and runnable and well tested demo app, please find all the aidl and related model class in it.
The **README.md** and **doc/KeyboardShotcuts.md** are detailed intro documents.

Appendix C: FAQ

Q: My debug and release builds have different signatures. Which should I provide?
A: Provide both. During development/testing, your debug signature will be used. For production, your release signature will be used. Both can be registered in the allowlist.

Q: What happens if I need to change my signing key?
A: You must notify the ADS team before the change. We'll update the allowlist to include both old and new signatures during a transition period. After all devices are updated, the old signature can be removed. Please aware that the signature change will impact your App OTA as well since the OTA only available between 2 release with a same signature.

Q: What if signature verification fails?
A: Remote Management will reject all API calls with an `UNAUTHORIZED` error code. Check that:

- Your package name matches the allowlist entry
- Your signature matches exactly (including correct format)
- Your app is properly signed (not using Android's default debug key for production)

Q: Does the signature verification work offline?
A: Yes, after the first successful verification. The verification result is cached locally with a TTL (e.g. default 24 hours). Only the initial verification or cache expiry requires network connectivity.

Q: What format should timestamps be in?
A: All timestamps use ISO 8601 format with timezone: `"2025-10-11T23:01:00.000Z"`. This is human-friendly and machine-parseable.

Q: Are memory and storage values in bytes or MB?
A: All values are in bytes to align with the REST API. Convert to MB by dividing by `1024 * 1024`. Example: `560029696 bytes = 534 MB`.

Q: Why do HDMI CEC commands sometimes not work?
A: CEC support varies by display. Not all displays support all CEC commands. Always:

- Check `canControl` flag in status responses
- Use `refreshHdmiStatus()` to query current state
- Wait ~500ms after refresh for CEC response
- Prefer toggle operations over explicit set operations
- Handle code 501 (Not Implemented) gracefully

Q: How do I know if a feature is supported?
A: Use `isFeatureSupported(String feature)` API. Supported feature names: "HDMI_CEC", "SCREENSHOT", "WIFI_CONFIG", "CACHE_CLEAR", "OTA_UPDATE", "SCREEN_ROTATION", "SCREEN_RECORD"

Q: What happens if I exceed the rate limit?
A: You'll receive a response with code 429 (Too Many Requests). Implement exponential backoff and retry logic. See Appendix B for rate limit details.

Q: How do I handle asynchronous CEC operations?
A: CEC status updates are asynchronous. Call `refreshHdmiStatus()`, wait 500ms, then call `getHdmiStatus()` to get updated power/audio state. See Appendix B.3 for examples.

Q: Can I call APIs on the main thread?
A: No. Always use background threads (AsyncTask, Executor, or Coroutines) for all API calls. AIDL calls are synchronous and will block the calling thread.

Q: Where can I find complete code examples?
A: See Appendix B: AIDL Interface Definitions for:

- Complete AIDL definitions
- Full implementation examples
- Service connection patterns
- Error handling examples
- Best practices

Q: How do I test the flow when a user is requested to allow *Device Management API?*
A: Just change your application's package name (**applicationId** in **build.gradle**) and then sideload it again (using adb install yourapk). This will allow you to experience the dynamic permission request process, as the system will treat it as a new app without pre-granted permissions.

Q: How do I handle Fire OS devices that don't include the ADS API? Should I provide two separate builds, one with the ADS API for Signage Sticks and one without it for Fire TV Sticks, or is it acceptable to ship a single app that checks for the ADS API at runtime and falls back to the current behavior if it's not available?
A: You can check if the Remote Management API is available at runtime using a helper method like this:

```
// Service details for binding
private static final String REMOTE_API_SERVICE_PACKAGE = "com.amazon.ads.ema.sys";
private static final String REMOTE_API_SERVICE_CLASS =
        "com.amazon.ads.ema.service.DeviceManagementService";

public boolean isServiceAvailable(Context context) {
    PackageManager pm = context.getPackageManager();
    Intent intent = new Intent();
```

```
    intent.setClassName(REMOTE_API_SERVICE_PACKAGE, REMOTE_API_SERVICE_CLASS);

    List<ResolveInfo> services = pm.queryIntentServices(intent, 0);

    return !services.isEmpty();
}
```

This allows you to detect whether the device has the Remote Management API and gracefully fall back to your current behavior if it's not available. This approach is much simpler than maintaining two separate builds and will work seamlessly across different Fire OS versions.