Manager for Android Customer API Key Usage Guide

Table of Contents

1. Overview
2. Getting Started
3. Authentication
4. API Endpoints
5. Request Examples
6. Response Formats
7. Error Handling
8. Security Considerations
9. Best Practices
10. Code Examples
11. Troubleshooting

1. Overview

The Manager for Android Customer API provides programmatic access to enterprise mobility management features using Customer API keys. This RESTful API follows the JSON:API specification and enables customers to manage devices, profiles, applications, and other EMM resources programmatically.

Key Features

- Full CRUD operations on EMM resources
- JSON:API compliant responses
- Multi-tenant architecture with namespace isolation
- Super Administrator permissions for API key users
- Comprehensive device and profile management
- Real-time device status monitoring

2. Getting Started

Prerequisites

- Valid Manager for Android customer account
- Customer API key (contact your Manager for Android administrator)
- Customer namespace identifier
- HTTPS-capable client application

API Base URL

Production: `https://your-instance.manager-for-android.jamfcloud.com`

3. Authentication

Required Headers

Every API request must include these mandatory headers:

Header	Description	Example
`WizyEMM-Customer-API-Key`	Your customer API key	`abc123def456ghi789`
`WizyEMM-Namespace`	Your customer namespace	`acme-corp`
`Content-Type`	Request content type	`application/vnd.api+json`

Authentication Flow

1. **API Key Validation**: The system validates your API key against the provided namespace
2. **Federation Context**: Sets up the customer federation context for data isolation
3. **Role Assignment**: Grants Super Administrator permissions for the session
4. **Request Processing**: Processes the API request within the customer's scope

> **Note:** Customer API keys provide Super Administrator permissions, giving full access to all resources within your customer namespace.

4. API Endpoints

Manager for Android provides two API versions with different capabilities and frameworks:

Version 1 Endpoints (`/api/v1/`) - CRNK Framework

Resource	Endpoint	Description
Devices	`/api/v1/devices`	Manage enrolled devices
Profiles	`/api/v1/profiles`	Device configuration profiles
Applications	`/api/v1/applications`	Manage applications
Play Store Apps	`/api/v1/playstoreapps`	Play Store application management
System Apps	`/api/v1/systemapps`	System application management
Administrators	`/api/v1/administrators`	Admin account management
Groups	`/api/v1/device-groups`	Device grouping (deprecated, use v2 labels)
Users	`/api/v1/users`	End user management (deprecated, use v2 endusers)
Commands	`/api/v1/commands`	Device command management
Customers	`/api/v1/customers`	Customer account management
Events	`/api/v1/events`	Event logging
Geolocations	`/api/v1/geolocations`	Device location data
Geofences	`/api/v1/geofences`	Geofence management
Signin Details	`/api/v1/signin-details`	Enrollment signin details
Settings	`/api/v1/settings`	Customer settings

Resource	Endpoint	Description
WiFi Networks	`/api/v1/wifi-networks`	WiFi network configurations
QR Codes	`/api/v1/qrcodes`	QR code enrollment
Register Devices	`/api/v1/register-devices`	Bulk device registration
Device Actions	`/api/v1/device-actions`	Device action management
Device Models	`/api/v1/device-models`	Device model information
Application Tracks	`/api/v1/application-tracks`	Application track management
Managed Configurations	`/api/v1/managed-configurations`	App configuration management
App Permissions	`/api/v1/app-permissions`	Application permissions
App States	`/api/v1/appstates`	Application state reporting
RMAs	`/api/v1/rmas`	Return Merchandise Authorization
Zebra Deployments	`/api/v1/zebra-deployments`	Zebra device deployments
Zebra Updates	`/api/v1/zebra-updates`	Zebra update management
Application Reports	`/api/v1/application-reports`	Application usage reports
Non-Compliance Details	`/api/v1/non-compliance-details`	Device compliance issues

Version 2 Endpoints (`/api/v2/`) - Elide Framework

Resource	Endpoint	Description	V2 Features
Devices	`/api/v2/devices`	Enhanced device management	Advanced filtering, relationships
Profiles	`/api/v2/profiles`	Configuration profiles	Enhanced profile management
Labels	`/api/v2/labels`	Device grouping and labeling	Replaces v1 groups

Resource	Endpoint	Description	V2 Features
Read-Only Labels	`/api/v2/readonlylabels`	Read-only label views	New in v2
End Users	`/api/v2/endusers`	End user management	Replaces v1 users
Events	`/api/v2/events`	Event logging and tracking	New in v2
Coordinates	`/api/v2/coordinates`	Device coordinates/location	New in v2
Read-Only Coordinates	`/api/v2/readonlycoordinates`	Read-only coordinate views	New in v2
Geofences	`/api/v2/geofences`	Geofencing management	Enhanced geofencing
Folders	`/api/v2/folders`	Organizational hierarchy	New in v2
WiFi Networks	`/api/v2/wifi-networks`	WiFi configurations	Enhanced WiFi management
Managed Apps	`/api/v2/managedapps`	Application entities	Enhanced app management
App States	`/api/v2/appstates`	Application state tracking	Enhanced state tracking
App Logs	`/api/v2/applogs`	Application logs	New in v2
Application Reports	`/api/v2/applicationreports`	Application usage reporting	Enhanced reporting
Usage Analytics	`/api/v2/usage`	Usage statistics and metrics	New in v2
Compliance	`/api/v2/noncompliancedetails`	Device compliance tracking	Enhanced compliance
Settings	`/api/v2/settings`	System configuration	Enhanced settings

Resource	Endpoint	Description	V2 Features
Customers	`/api/v2/customers`	Customer management	Enhanced customer data
Security Issues	`/api/v2/securityissues`	Security issue tracking	New in v2
Battery Info	`/api/v2/batteryinfos`	Battery information	New in v2
iOS Device Applications	`/api/v2/iosdevice_applications`	iOS device applications	New in v2
iOS Configuration Profiles	`/api/v2/iosdevice_configurationprofiles`	iOS device configuration profiles	New in v2
iOS Provisioning Profiles	`/api/v2/iosdevice_provisioningprofiles`	iOS device provisioning profiles	New in v2
iOS Config Profile Templates	`/api/v2/iosconfigurationprofiles`	iOS configuration profile templates	New in v2
iOS Provisioning Templates	`/api/v2/iosprovisioningprofiles`	iOS provisioning profile templates	New in v2
Enrollment Workflows	`/api/v2/enrollmentworkflows`	Enrollment workflow management	New in v2
QR Codes	`/api/v2/qrcodes`	QR code management	Enhanced QR features
OEM Integration	`/api/v2/oemintegration`	OEM integration settings	New in v2
Packages	`/api/v2/packages`	Package management	New in v2
VisibilityIQ	`/api/v2/visibilityiq`	VisibilityIQ integration	New in v2

Zebra-Specific V2 Endpoints

Resource	Endpoint	Description

Resource	Endpoint	Description
Zebra Devices	`/api/v2/zebradevices`	Zebra-specific device management
Zebra Artifacts	`/api/v2/zebraartifacts`	Deployment artifacts
Zebra Deployments	`/api/v2/zebradeployments`	Deployment management
SQL Device Deployment Details	`/api/v2/sqldevicedeploymentdetails`	Zebra deployment details

Specialized Endpoints

- **Zebra Integration**: `/api/zebra/*`
- **Zoho Integration**: `/api/zoho/*`
- **Data Export**: `/api/export/*`
- **Analytics**: `/api/metrics/*`
- **OEM Features**: `/api/oem/*`

> **API Version Recommendation:** Use API v2 for new integrations. V1 endpoints are maintained for backward compatibility but v2 provides enhanced features and better performance.

5. Request Examples

API v1 Examples

Get All Devices (v1)

```
GET /api/v1/devices HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace
```

Create a New Profile (v1)

```
POST /api/v1/profiles HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace

{
  "data": {
    "type": "profiles",
    "attributes": {
      "name": "Corporate Security Profile",
      "description": "Standard corporate security settings",
      "managementMode": "MANAGED"
    }
  }
}
```

API v2 Examples

Get All Devices with Enhanced Filtering (v2)

```
GET /api/v2/devices?filter=state=='REGISTERED'&include=labels,enduser HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace
```

Create a Geofence (v2 only)

```
POST /api/v2/geofences HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace

{
  "data": {
    "type": "geofences",
    "attributes": {
      "name": "Corporate Headquarters",
      "latitude": 37.7749,
      "longitude": -122.4194,
      "radius": 100
    }
  }
}
```

Get Device Events (v2 only)

```
GET /api/v2/events?filter=deviceId=='550e8400-e29b-41d4-a716-446655440000' HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace
```

Update Device Labels using V2

```
PATCH /api/v2/devices/550e8400-e29b-41d4-a716-446655440000 HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace

{
  "data": {
    "type": "devices",
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "relationships": {
      "labels": {
        "data": [
          {"type": "labels", "id": "label-uuid-1"},
          {"type": "labels", "id": "label-uuid-2"}
        ]
      }
    }
  }
}
```

Create an End User (v2 only)

```
POST /api/v2/endusers HTTP/1.1
Host: your-instance.manager-for-android.jamfcloud.com
Content-Type: application/vnd.api+json
WizyEMM-Customer-API-Key: your-api-key-here
WizyEMM-Namespace: your-namespace

{
  "data": {
    "type": "endusers",
    "attributes": {
      "email": "john.doe@company.com",
      "firstName": "John",
      "lastName": "Doe"
    }
  }
}
```

6. Response Formats

Successful Response (200 OK)

```
{
  "data": [
    {
      "type": "devices",
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "attributes": {
        "name": "John's iPhone",
        "serialNumber": "ABC123DEF456",
        "state": "REGISTERED",
        "deviceType": "IOS",
        "lastSeen": "2025-08-28T15:30:00Z"
      },
      "relationships": {
        "profile": {
          "data": {
            "type": "profiles",
            "id": "profile-uuid-here"
          }
        }
      }
    }
  ],
  "meta": {
    "totalCount": 1
  }
}
```

7. Error Handling

Common Error Codes

HTTP Status	Error Code	Description
401	UNAUTHORIZED	Invalid or missing API key
403	FORBIDDEN	Access denied to resource
404	NOT_FOUND	Resource not found
422	VALIDATION_ERROR	Invalid request data
500	INTERNAL_ERROR	Server error

Authentication Error Example

```
{
  "errors": [
    {
      "status": "401",
      "code": "CUSTOMER_KEY_INVALID",
      "title": "Invalid Customer API Key",
      "detail": "Customer key does not exist or does not belong to the right customer"
    }
  ]
}
```

8. Security Considerations

Important: Customer API keys provide Super Administrator permissions. Handle them with extreme care.

Access Control

- **Super Administrator Permissions**: Full administrative access within your namespace
- **Namespace Isolation**: Cannot access other customers' data
- **Federation Security**: Strict multi-tenant separation

Best Security Practices

- Store API keys in environment variables or secure key management systems
- Never expose API keys in client-side code or version control
- Use HTTPS exclusively for all API communications
- Implement request signing for additional security layers
- Monitor API usage patterns for anomalies
- Rotate API keys regularly (contact administrator)
- Restrict API key access to authorized systems only

9. Best Practices

Request Optimization

- Use pagination for large datasets to improve performance
- Implement proper caching strategies for frequently accessed data
- Include specific field selections when possible to reduce payload size
- Use bulk operations when available for multiple resource updates

Error Handling

- Always check HTTP status codes before processing responses
- Parse error responses for detailed information
- Implement retry logic with exponential backoff for transient errors
- Log API errors for debugging and monitoring

Rate Limiting

- Implement client-side rate limiting to avoid overwhelming the API
- Monitor API usage patterns and quotas
- Use efficient polling intervals for status checks

10. Code Examples

Python Example

```python
import requests
import json
import os

class ManagerForAndroidClient:
    def __init__(self, base_url, api_key, namespace):
        self.base_url = base_url
        self.headers = {
            'Content-Type': 'application/vnd.api+json',
            'WizyEMM-Customer-API-Key': api_key,
            'WizyEMM-Namespace': namespace
        }

    def get_devices(self, page=1, size=50):
        """Get paginated list of devices"""
        params = {'page[number]': page, 'page[size]': size}
        response = requests.get(
            f"{self.base_url}/api/v1/devices",
            headers=self.headers,
            params=params
        )
        response.raise_for_status()
        return response.json()

    def create_profile(self, name, description, management_mode="MANAGED"):
        """Create a new device profile"""
        data = {
            "data": {
                "type": "profiles",
                "attributes": {
                    "name": name,
                    "description": description,
                    "managementMode": management_mode
                }
            }
        }
        response = requests.post(
            f"{self.base_url}/api/v1/profiles",
            headers=self.headers,
            json=data
        )
        response.raise_for_status()
        return response.json()

# Usage
client = ManagerForAndroidClient(
    base_url=os.getenv('MANAGER_BASE_URL'),
    api_key=os.getenv('MANAGER_API_KEY'),
    namespace=os.getenv('MANAGER_NAMESPACE')
)

devices = client.get_devices()
print(f"Found {devices['meta']['totalCount']} devices")
```

JavaScript/Node.js Example

```
const axios = require('axios');

class WizyEMMClient {
    constructor(baseUrl, apiKey, namespace) {
        this.baseUrl = baseUrl;
        this.headers = {
            'Content-Type': 'application/vnd.api+json',
            'WizyEMM-Customer-API-Key': apiKey,
            'WizyEMM-Namespace': namespace
        };
    }

    async getDevices(page = 1, size = 50) {
        try {
            const response = await axios.get(
                `${this.baseUrl}/api/v1/devices`,
                {
                    headers: this.headers,
                    params: { 'page[number]': page, 'page[size]': size }
                }
            );
            return response.data;
        } catch (error) {
            console.error('API Error:', error.response?.data || error.message);
            throw error;
        }
    }
}

// Usage
const client = new ManagerForAndroidClient(
    process.env.MANAGER_BASE_URL,
    process.env.MANAGER_API_KEY,
    process.env.MANAGER_NAMESPACE
);

client.getDevices()
    .then(devices => console.log(devices))
    .catch(error => console.error(error));
```

11. Troubleshooting

Common Issues and Solutions

1. Authentication Failed (401)

Problem: Invalid API key or namespace

Solutions:

- Verify API key is correct and active
- Ensure namespace matches your customer account exactly
- Check that API key hasn't expired
- Confirm you're using the correct API endpoint URL

2. Access Denied (403)

Problem: Insufficient permissions or wrong namespace

Solutions:

- Verify you're accessing resources within your namespace
- Contact administrator to verify API key permissions
- Check if the resource exists in your customer account

3. Validation Errors (422)

Problem: Invalid request data format

Solutions:

- Check JSON:API format compliance
- Validate all required fields are present
- Ensure data types match API specification
- Review request payload structure

Debug Checklist

1. **Verify Headers**: Ensure all required headers are present and correct
2. **Check Namespace**: Confirm namespace matches your customer account
3. **Validate JSON**: Ensure request body follows JSON:API format
4. **Test Connection**: Try simple GET request first
5. **Review Logs**: Check server logs for detailed error information
6. **Test Environment**: Verify you're connecting to the correct API endpoint

Getting Your API Key

> **Contact Information:** For API key requests, technical support, or additional documentation, contact your Manager for Android administrator or support team.

To request a Customer API key:

1. Contact your Manager for Android administrator
2. Provide the intended use case for the API key

3. Specify which systems will use the API key
4. Request your customer namespace identifier

Document Information

Version: 1.0 | Last Updated: August 2025

Generated from Manager for Android codebase analysis