

# **Native C/C++ Camera Application for Android (AOSP Vanilla – Ice Cream) Project Documentation**

**Done by:**

**Akhilsai Vodnala**

# Contents

<b>Native C/C++ Camera Application for Android (AOSP Vanilla – Ice Cream)</b> .....	1
<b>1. INTRODUCTION</b> .....	3
<b>2. Project Objectives</b> .....	4
<b>3. TECHNOLOGY STACK</b> .....	5
<b>4. Project Architecture Overview</b> .....	6
<b>5. Android Manifest Configuration</b> .....	7
<b>Permissions and Features</b> .....	7
<b>Application &amp; Activity</b> .....	7
<b>6. Native Build System (CMake)</b> .....	8
<b>CMakeLists.txt</b> .....	8
<b>7. Native Camera Implementation (native-lib.cpp)</b> .....	9
<b>7.1 Global State Management</b> .....	9
<b>7.2 Camera Initialization Flow</b> .....	9
<b>7.3 Camera Preview</b> .....	9
<b>7.4 Image Capture (Still JPEG)</b> .....	9
<b>7.5 Image Processing &amp; Saving</b> .....	10
<b>7.6 Restarting Preview</b> .....	10
<b>7.7 Camera Shutdown</b> .....	10
<b>8. JNI Interface Design</b> .....	11
<b>9. UI Layout (activity_main.xml)</b> .....	12
<b>Components</b> .....	12
<b>10. Gradle Configuration</b> .....	13
<b>App-Level Gradle</b> .....	13
<b>Root Gradle</b> .....	13
<b>11. Key Learnings &amp; Highlights</b> .....	14
<b>12. Future Enhancements</b> .....	15
<b>13. Conclusion</b> .....	16

# 1. INTRODUCTION

This project implements a **native Camera application** for Android using C/C++ (**NDK**) with minimal Java/Kotlin involvement. The application is designed for **AOSP Vanilla (Ice Cream)** builds and demonstrates direct usage of **Android Camera2 NDK APIs** for camera preview and still image capture.

Unlike typical Android camera apps that rely on CameraX or Camera2 Java APIs, this project directly interfaces with the **native camera stack**, making it suitable for: - AOSP / custom ROM development - System-level or near-system-level applications - Embedded and platform-level Android development - Understanding Android OS internals and HAL interaction

## 2. Project Objectives

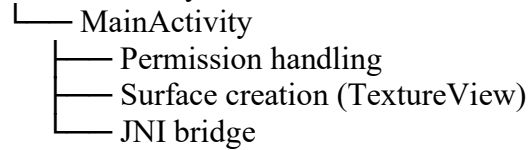
- Build a native camera application using **Android NDK**
- Use **Camera2 NDK (camera2ndk)** APIs instead of Java Camera APIs
- Render camera preview using **ANativeWindow + TextureView**
- Capture JPEG images using **AIImageReader**
- Save captured images directly to the Android **MediaStore (Gallery)**
- Maintain a thin Java layer only for lifecycle and permissions

### 3. TECHNOLOGY STACK

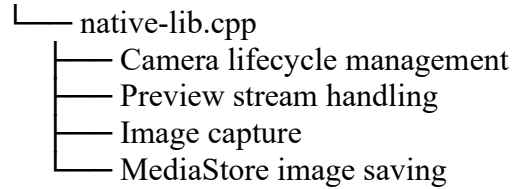
Layer	Technology
UI	XML Layout + TextureView
App Layer	Minimal Java Activity
Native Layer	C++ (NDK)
Camera API	Camera2 NDK (camera2ndk)
Media API	Media NDK (mediandk)
Build System	Gradle + CMake
Android Version	minSdk 26, targetSdk 36

## 4. Project Architecture Overview

### Java/Kotlin Layer



### JNI Interface



### Android Framework



The Java layer is intentionally minimal and delegates all camera operations to the native layer through JNI.

## 5. Android Manifest Configuration

### Permissions and Features

```
<uses-permission android:name="android.permission.CAMERA" />  
<uses-feature  
  android:name="android.hardware.camera"  
  android:required="true" />
```

- Declares camera permission
- Ensures the device has camera hardware

### Application & Activity

- Single launcher activity: MainActivity
- Portrait orientation enforced
- No background services or extra components

## 6. Native Build System (CMake)

### CMakeLists.txt

The native library is built as a **shared library** and linked against required Android NDK components.

Key points: - Builds native-lib.cpp into libnativecamera.so - Links against: - android (native window support) - camera2ndk (Camera2 native APIs) - mediandk (ImageReader & media handling) - log (Android logging)

This setup enables direct interaction with the Android camera framework from C++.

## 7. Native Camera Implementation (native-lib.cpp)

### 7.1 Global State Management

The native layer maintains global references for: - JavaVM and Activity (for JNI calls) - Camera objects (ACameraManager, ACameraDevice, ACameraCaptureSession) - Preview and capture surfaces (ANativeWindow) - Image reader (AImageReader)

These objects persist across JNI calls and are carefully created and destroyed during lifecycle events.

---

### 7.2 Camera Initialization Flow

1. **Permission granted** from Java layer
  2. **Preview surface available** (TextureView)
  3. startCamera() is invoked
  4. Camera ID is obtained from ACameraManager
  5. Camera device is opened
  6. Capture session is created with:
    - Preview output surface
    - ImageReader output surface
  7. Repeating preview request starts
- 

### 7.3 Camera Preview

- Preview frames are streamed to a TextureView
- ANativeWindow\_fromSurface() converts Java surface to native window
- Preview uses TEMPLATE\_PREVIEW

This provides low-latency camera preview fully managed in native code.

---

### 7.4 Image Capture (Still JPEG)

- Triggered via JNI call from PHOTO button
- Uses TEMPLATE\_STILL\_CAPTURE
- Output target: AImageReader configured for JPEG
- JPEG orientation is explicitly set

Captured images are delivered asynchronously via onImageAvailable() callback.

## 7.5 Image Processing & Saving

- `AlImageReader_acquireNextImage()` retrieves the JPEG image
- Raw JPEG buffer is extracted using `AlImage_getPlaneData()`
- Image is saved using Android **MediaStore API** via JNI:
  - Inserts image entry into gallery
  - Writes JPEG bytes to output stream

This ensures images appear immediately in the system Gallery app.

---

## 7.6 Restarting Preview

After a still capture: - The repeating preview request is restarted - This mimics standard camera app behavior

---

## 7.7 Camera Shutdown

Handled in `stopCamera()`: - Capture session closed - Camera device closed - Camera manager destroyed

Invoked when the app goes to background or activity pauses.

## 8. JNI Interface Design

The following JNI methods bridge Java and native layers:

JNI Method	Purpose
<code>nativeSetSurface()</code>	Passes preview surface to native code
<code>nativeOnPermissionResult()</code>	Notifies permission state
<code>nativeOnResume()</code>	Restarts camera
<code>nativeOnPause()</code>	Stops camera
<code>nativeCaptureImage()</code>	Triggers still capture

JNI is used strictly as a bridge — no camera logic exists in Java.

## 9. UI Layout (activity\_main.xml)

### Components

- TextureView: Fullscreen camera preview
- Bottom control bar:
  - PHOTO button (implemented)
  - VIDEO button (placeholder for future extension)

The layout overlays controls on top of the camera preview.

## 10. Gradle Configuration

### App-Level Gradle

- Uses Kotlin DSL
- Enables externalNativeBuild (CMake)
- C++17 standard enabled
- minSdk 26 (Camera2 NDK requirement)
- targetSdk 36 (AOSP Ice Cream)

### Root Gradle

- Centralized plugin management
- Clean modular project structure

## **11. Key Learnings & Highlights**

- Direct interaction with Android Camera Service via NDK
- Understanding of Camera2 pipeline at native level
- JNI lifecycle and thread attachment handling
- MediaStore integration from native code
- Native preview rendering using ANativeWindow

## **12. Future Enhancements**

- Implement video recording using MediaCodec / MediaMuxer
- Add camera parameter controls (AF, AE, AWB)
- Support multiple cameras
- Improve error recovery and robustness
- Migrate to system app for deeper AOSP integration

## 14. Conclusion

This project demonstrates a **true native Android camera application** built using C/C++ and the Camera2 NDK. It serves as a strong foundation for AOSP development, embedded Android systems, and engineers aiming to understand Android's lower-level camera architecture beyond standard SDK abstractions.