**Camera API**

**Table of Contents :**

* Introduction
* Manifest declaration
* Building a Camera APP

### Detecting camera hardware

### Accessing cameras

### Checking camera features

### Creating a preview class

### Capturing pictures

## Implementing the HAL

### 

**Introduction**

The Android framework includes support for various cameras and camera features available on devices, allowing to capture pictures and videos in applications. This document discusses a quick, simple approach to image and video capture and outlines an advanced approach for creating custom camera experiences for end users.

* The Android framework supports capturing images and video through the [android.hardware.camera2](https://developer.android.com/reference/android/hardware/camera2/package-summary) API
* [android.hardware.camera2](https://developer.android.com/reference/android/hardware/camera2/package-summary)

This package is the primary API for controlling device cameras. It can be used to take pictures or videos when we are building a camera application.

* [SurfaceView](https://developer.android.com/reference/android/view/SurfaceView)

This class is used to present a live camera preview to the user.

* [MediaRecorder](https://developer.android.com/reference/android/media/MediaRecorder)

This class is used to record video from the camera.

## Manifest declarations

Before starting development on our application with the Camera API, we should make sure the manifest has the appropriate declarations to allow use of camera hardware and other related features.

* **Camera Permission**- The application must request permission to use a device camera.

<uses-permission android:name="android.permission.CAMERA" />

* **Camera Features** - The application must also declare use of camera features, for example:

<uses-feature android:name="android.hardware.camera" />

* android.hardware.camera.flash

The app uses the flash feature that the device's camera supports. By using this feature, an app implies that it also uses the android.hardware.camera feature, unless this parent feature is declared with android:required="false".

* android.hardware.camera.front

The app uses the device's front-facing camera.

* android.hardware.camera.autofocus

The app uses the autofocus feature that the device's camera supports.

* android.hardware.camera

The app uses the device's back-facing camera. Devices with only a front-facing camera do not list this feature, so use the android.hardware.camera.any feature instead if our app can communicate with any camera, regardless of which direction the camera faces.

* **Storage Permission** - The application can save images or videos to the device's external storage (SD Card) if it targets Android 10 (API level 29) or lower and specifies the following in the manifest.

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

* **Audio Recording Permission** - For recording audio with video capture, our application must request the audio capture permission.

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

* **Location Permission** - If our application tags images with GPS location information, must request the ACCESS\_FINE\_LOCATION permission. Note that, if our app targets Android 5.0 (API level 21) or higher, also need to declare that our app uses the device's GPS:

## Building a camera app

The general steps for creating a custom camera interface for our application are as follows:

* **Detect and Access Camera** - Create code to check for the existence of cameras and request access.
* **Create a Preview Class** – For users to effectively take pictures or video, they must be able to see what the device camera sees. A camera preview class is a [SurfaceView](https://developer.android.com/reference/android/view/SurfaceView) that can display the live image data coming from a camera, so users can frame and capture a picture or video.

* **Setup Listeners for Capture** - Connect listeners for interface controls to start image or video capture in response to user actions, such as pressing a button.
* **Capture and Save Files** - Setup the code for capturing pictures or videos and saving the output.
* **Release the Camera** - After using the camera, the application must properly release it for use by other applications.

### Detecting camera hardware

If the application does not specifically require a camera using a manifest declaration, we should check to see if a camera is available at runtime

\* Check if this device has a camera \*/  
private boolean checkCameraHardware(Context context) {  
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE\_CAMERA)){  
        // this device has a camera  
        return true;  
    } else {  
        // no camera on this device  
        return false;  
    }

Android devices can have multiple cameras, for example a back-facing camera for photography and a front-facing camera for video calls. Android 2.3 (API Level 9) and later allows us to check the number of cameras available on a device using the [Camera.getNumberOfCameras()](https://developer.android.com/reference/android/hardware/Camera" \l "getNumberOfCameras()) method.Returns the number of physical cameras available on this device. The return value of this method might change dynamically if the device supports external cameras and an external camera is connected or disconnected

public static class CameraInfo {

/\* The facing of the camera is opposite to that of the screen.\*/

public static final int CAMERA\_FACING\_BACK = 0;

/\* The facing of the camera is the same as that of the screen.\*/

public static final int CAMERA\_FACING\_FRONT = 1;

### Accessing cameras

If we have determined that the device on which the application is running has a camera, we must request to access it by getting an instance of [Camera](https://developer.android.com/reference/android/hardware/Camera)

/\*\* A safe way to get an instance of the Camera object. \*/  
public static Camera getCameraInstance(){  
    Camera c = null;  
    try {  
        c = Camera.open(); // attempt to get a Camera instance  
    }  
    catch (Exception e){  
        // Camera is not available (in use or does not exist)  
    }  
    return c; // returns null if camera is unavailable  
}

here example code above will access the first primary camera i.e., back-facing camera on a device with more than one camera.

### Checking camera features

Use the [Camera.getCameraInfo()](https://developer.android.com/reference/android/hardware/Camera" \l "getCameraInfo(int, android.hardware.Camera.CameraInfo)) to determine if a camera is on the front or back of the device, and the orientation of the image.

if (info.facing == CameraInfo.CAMERA\_FACING\_FRONT) {

\* rotation = (info.orientation - orientation + 360) % 360;

\* } else { // back-facing camera

\* rotation = (info.orientation + orientation) % 360;

\* }

Once we obtain access to a camera, we can get further information about its capabilities using the [Camera.getParameters()](https://developer.android.com/reference/android/hardware/Camera" \l "getParameters()) method

## Camera features are :

Android supports a wide array of camera features we can control with our camera application, such as picture format, flash mode, focus settings, and many more.

|  |  |
| --- | --- |
| Feature | Description |
| [Face Detection](https://developer.android.com/guide/topics/media/camera" \l "face-detection) | Identify human faces within a picture and use them for focus, metering and white balance |
| [Focus Areas](https://developer.android.com/guide/topics/media/camera" \l "metering-focus-areas) | Set one or more areas within an image to use for focus |
| [White Balance Lock](https://developer.android.com/reference/android/hardware/Camera.Parameters" \l "setAutoWhiteBalanceLock(boolean)) | Stop or start automatic white balance adjustments |
|  |  |
| [Zoom](https://developer.android.com/reference/android/hardware/Camera.Parameters" \l "setZoom(int)) | Set image magnification |
| [Flash Mode](https://developer.android.com/reference/android/hardware/Camera.Parameters" \l "setFlashMode(java.lang.String)) | Turn flash on, off, or use automatic setting |
| [Color Effects](https://developer.android.com/reference/android/hardware/Camera.Parameters" \l "setColorEffect(java.lang.String)) | Apply a color effect to the captured image such as black and white, sepia tone or negative. |

// get Camera parameters  
Camera.Parameters params = camera.getParameters();  
// set the focus mode  
params.setFocusMode(Camera.Parameters.FOCUS\_MODE\_AUTO);  
// set Camera parameters  
camera.setParameters(params);

The application must start the face detection function each time we start (or restart) the camera preview. Create a method for starting face detection so we can call it as needed, as shown in the example code below.

public void startFaceDetection(){  
    // Try starting Face Detection  
    Camera.Parameters params = mCamera.getParameters();  
  
    // start face detection only \*after\* preview has started  
    if (params.getMaxNumDetectedFaces() > 0){  
        // camera supports face detection, so can start it:  
        mCamera.startFaceDetection();  
    }  
}

### Creating a preview class

For users to effectively take pictures or video, they must be able to see what the device camera sees. A camera preview class is a [SurfaceView](https://developer.android.com/reference/android/view/SurfaceView) that can display the live image data coming from a camera, so users can frame and capture a picture or video.

public void surfaceCreated(SurfaceHolder holder) {  
        // The Surface has been created, now tell the camera where to draw the preview.  
        try {  
            mCamera.setPreviewDisplay(holder);  
            mCamera.startPreview();  
        } catch (IOException e) {  
            Log.d(TAG, "Error setting camera preview: " + e.getMessage());  
        }  
    }

### Capturing pictures

Once we have built a preview class and a view layout in which to display it, we are ready to start capturing images with our application. In the application code, we must set up listeners for the user interface controls to respond to a user action by taking a picture.

In order to retrieve a picture, use the [Camera.takePicture()](https://developer.android.com/reference/android/hardware/Camera" \l "takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) method. This method takes three parameters which receive data from the camera...

In order to receive data in a JPEG format, we must implement an [Camera.PictureCallback](https://developer.android.com/reference/android/hardware/Camera.PictureCallback) interface to receive the image data and write it to a file.

The following code shows a basic implementation of the [Camera.PictureCallback](https://developer.android.com/reference/android/hardware/Camera.PictureCallback) interface to save an image received from the camera.

/ Add a listener to the Capture button  
Button captureButton = (Button) findViewById(R.id.button\_capture);  
captureButton.setOnClickListener(  
    new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            // get an image from the camera  
            mCamera.takePicture(null, null, picture);  
        }

Cameras are a resource that is shared by applications on a device. The application can make use of the camera after getting an instance of [Camera](https://developer.android.com/reference/android/hardware/Camera), and we must be particularly careful to release the camera object when the application stops using it,

private void releaseCamera(){  
        if (mCamera != null){  
            mCamera.release();        // release the camera for other applications  
            mCamera = null;  
        }  
    }

## Saving media files

Media files created by users such as pictures and videos should be saved to a device's external storage directory (SD Card) to conserve system space and to allow users to access these files without their device. There are many possible directory locations to save media files on a device, however there are only two standard locations we should consider as a developer:

[Environment.getExternalStoragePublicDirectory](https://developer.android.com/reference/android/os/Environment" \l "getExternalStoragePublicDirectory(java.lang.String))([Environment.DIRECTORY\_PICTURES](https://developer.android.com/reference/android/os/Environment" \l "DIRECTORY_PICTURES)) –

This method returns the standard, shared and recommended location for saving pictures and videos. This directory is shared (public), so other applications can easily discover, read, change and delete files saved in this location.  If the application is uninstalled by the user, media files saved to this location will not be removed

\*\* Create a File for saving an image or video \*/  
private static File getOutputMediaFile(int type){  
    // To be safe, you should check that the SDCard is mounted  
    // using Environment.getExternalStorageState() before doing this.  
  
    File mediaStorageDir = new File(Environment.getExternalStoragePublicDirectory(  
              Environment.DIRECTORY\_PICTURES), "MyCameraApp");  
    // This location works best if you want the created images to be shared  
    // between applications and persist after your app has been uninstalled.  
  
    // Create the storage directory if it does not exist  
    if (! mediaStorageDir.exists()){  
        if (! mediaStorageDir.mkdirs()){  
            Log.d("MyCameraApp", "failed to create directory");  
            return null;  
        }  
    }

## Implementing the HAL

Android's camera Hardware Abstraction Layer (HAL) connects the higher level camera framework APIs in [android.hardware.camera2](https://developer.android.com/reference/android/hardware/camera2/package-summary) to the underlying camera driver and hardware.

The HAL sits between the camera driver and the higher-level Android framework and defines an interface .So apps can correctly operate the camera hardware.

* The [HIDL](https://source.android.com/devices/architecture/hidl) interfaces for the Camera HAL are defined in [hardware/interfaces/camera](https://android.googlesource.com/platform/hardware/interfaces/+/master/camera/).

A typical binderized HAL must implement the following HIDL interfaces:

* [ICameraProvider](https://android.googlesource.com/platform/hardware/interfaces/+/refs/heads/master/camera/provider/2.4/ICameraProvider.hal): For enumerating individual devices and managing their status and

The framework will try to establish a connection.

* [ICameraDevice](https://android.googlesource.com/platform/hardware/interfaces/+/refs/heads/master/camera/device/3.2/ICameraDevice.hal): The camera device interface.
* [ICameraDeviceSession](https://android.googlesource.com/platform/hardware/interfaces/+/refs/heads/master/camera/device/3.2/ICameraDeviceSession.hal): The active camera device session interface. The framework calls ICameraDevice::open() to create a new active capture session ICameraDeviceSession.
* The HAL interface is defined in the

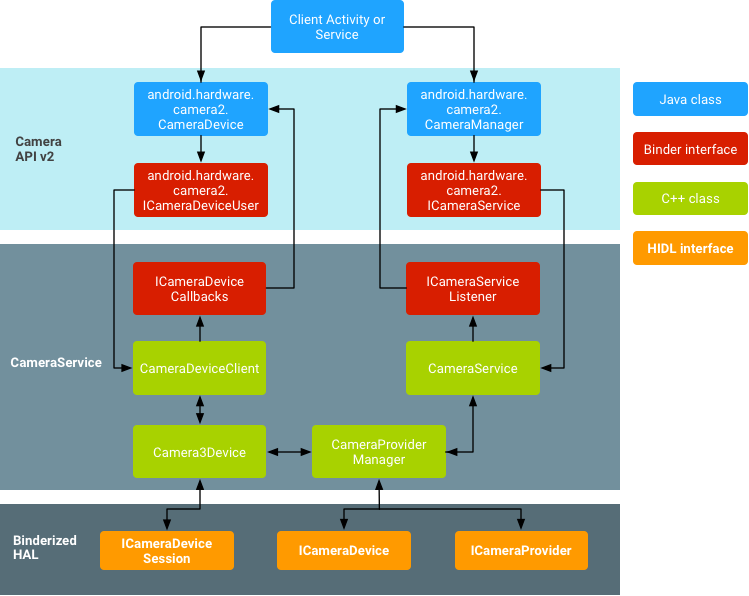
 hardware/libhardware/include/hardware/camera.h and hardware/libhardware/include/hardware/camera\_common.h header files.

camera\_common.h defines camera\_module, a standard structure to obtain general information about the camera, such as the camera ID and properties common to all cameras (that is, whether it is a front- or back-facing camera).

* HAL Implementation defined at hardware/qcom/camera/QCamera2/HAL
* For documentation on the camera parameters developers can set, refer to frameworks/av/include/camera/CameraParameters.h. These parameters are set with the function pointed to by int (\*set\_parameters)(struct camera\_device \*, const char \*parms) in the HAL.

Architecture

The following figure and list describe the HAL components.



Android camera architecture

Figure 1. Camera architecture