Camera

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

IN THIS DOCUMENT

Considerations
The Basics
Manifest Declarations
Using Existing Camera
- Image captu intent
- Video captur intent
- Receiving ca intent result

Building a Camera App
- Detecting ca hardware
- Accessing ca
- Checking car features
- Creating a pr class
- Placing previ layout
- Capturing pic
- Capturing vic
- Releasing th camera

Saving Media Files
Camera Features
- Checking fea availability
- Using camer features
- Metering and areas
- Face detecti
- Time lapse v

KEY CLASSES

`Camera`
`SurfaceView`
`MediaRecorder`
`Intent`

SEE ALSO

Media Playback
Data Storage

Considerations

Before enabling your application to use cameras on Android devices, you should consider a few questions about how your app intends to use this hardware feature.

- **Camera Requirement** - Is the use of a camera so important to your application that you do not want your application installed on a device that does not have a camera? If so, you should declare the camera requirement in your manifest.
- **Quick Picture or Customized Camera** - How will your application use the camera? Are you just interested in snapping a quick picture or video clip, or will your application provide a new way to use cameras? For a getting a quick snap or clip, consider Using Existing Camera Apps. For developing a customized camera feature, check out the Building a Camera App section.
- **Storage** - Are the images or videos your application generates intended to be only visible to your application or shared so that other applications such as Gallery or other media and social apps can use them? Do you want the pictures and videos to be available even if your application is uninstalled? Check out the Saving Media Files section to see how to implement these options.

The Basics

The Android framework supports capturing images and video through the `android.hardware.camera2` (/reference/android/hardware/camera2/package-summary.html) API or camera `Intent` (/reference/android/content/Intent.html). Here are the relevant classes:

`android.hardware.camera2`
: This package is the primary API for controlling device cameras. It can be used to take pictures or videos when you are building a camera application.

`Camera`
: This class is the older deprecated API for controlling device cameras.

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

`MediaRecorder`
: This class is used to record video from the camera.

`Intent`
: An intent action type of `MediaStore.ACTION_IMAGE_CAPTURE` or `MediaStore.ACTION_VIDEO_CAPTU` be used to capture images or videos without directly using the `Camera` object.

Manifest Declarations

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

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

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

> **Note:** If you are using the camera via an intent (#intents), your application does not need to request t
> permission.

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

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

For a list of camera features, see the manifest Features Reference (/guide/topics/manifest/uses-feature-element.html#hw-features).

Adding camera features to your manifest causes Google Play to prevent your application from being installed to devices that do not include a camera or do not support the camera features you specify. information about using feature-based filtering with Google Play, see Google Play and Feature-Basec Filtering (/guide/topics/manifest/uses-feature-element.html#market-feature-filtering).

If your application *can use* a camera or camera feature for proper operation, but does not *require* it, yc specify this in the manifest by including the `android:required` attribute, and setting it to `false`:

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

- **Storage Permission** - If your application saves images or videos to the device's external storage (SD (you must also specify this in the manifest.

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

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

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

- **Location Permission** - If your application tags images with GPS location information, you must reque location permission:

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

For more information about getting user location, see Location Strategies (/guide/topics/location/strategie

Using Existing Camera Apps

A quick way to enable taking pictures or videos in your application without a lot of extra code is to us `Intent (/reference/android/content/Intent.html)` to invoke an existing Android camera application. A came intent makes a request to capture a picture or video clip through an existing camera app and then ret control back to your application. This section shows you how to capture an image or video using this technique.

The procedure for invoking a camera intent follows these general steps:

1. **Compose a Camera Intent** - Create an `Intent` that requests an image or video, using one of thes types:

    - MediaStore.ACTION_IMAGE_CAPTURE - Intent action type for requesting an image from an existin
      camera application.
    - MediaStore.ACTION_VIDEO_CAPTURE - Intent action type for requesting a video from an existing
      application.
2. **Start the Camera Intent** - Use the startActivityForResult() method to execute the camera int
   After you start the intent, the Camera application user interface appears on the device screen
   user can take a picture or video.
3. **Receive the Intent Result** - Set up an onActivityResult() method in your application to receive
   callback and data from the camera intent. When the user finishes taking a picture or video (or
   the operation), the system calls this method.

Image capture intent

Capturing images using a camera intent is quick way to enable your application to take pictures with minimal coding. An image capture intent can include the following extra information:

- MediaStore.EXTRA_OUTPUT - This setting requires a Uri object specifying a path and file name where y to save the picture. This setting is optional but strongly recommended. If you do not specify this valu camera application saves the requested picture in the default location with a default name, specifie returned intent's Intent.getData() field.

The following example demonstrates how to construct a image capture intent and execute it. The getOutputMediaFileUri() method in this example refers to the sample code shown in Saving Media (#saving-media).

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private Uri fileUri;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // create Intent to take a picture and return control to the calling application
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE); // create a file to save the ima
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // set the image file name

    // start the image capture Intent
    startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
```

When the startActivityForResult() (/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) method is executed, see a camera application interface. After the user finishes taking a picture (or cancels the operation) user interface returns to your application, and you must intercept the onActivityResult() (/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) method to receive the the intent and continue your application execution. For information on how to receive the completed see Receiving camera intent result (#intent-receive).

Video capture intent

Capturing video using a camera intent is a quick way to enable your application to take videos with n coding. A video capture intent can include the following extra information:

- MediaStore.EXTRA_OUTPUT - This setting requires a Uri specifying a path and file name where you'd lil save the video. This setting is optional but strongly recommended. If you do not specify this value, th Camera application saves the requested video in the default location with a default name, specified returned intent's Intent.getData() field.
- MediaStore.EXTRA_VIDEO_QUALITY - This value can be 0 for lowest quality and smallest file size or 1 f

highest quality and larger file size.

- MediaStore.EXTRA_DURATION_LIMIT - Set this value to limit the length, in seconds, of the video being c
- MediaStore.EXTRA_SIZE_LIMIT - Set this value to limit the file size, in bytes, of the video being captur

The following example demonstrates how to construct a video capture intent and execute it. The getOutputMediaFileUri() method in this example refers to the sample code shown in Saving Media (#saving-media).

```
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
private Uri fileUri;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //create new Intent
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

    fileUri = getOutputMediaFileUri(MEDIA_TYPE_VIDEO);  // create a file to save the vi
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // set the image file name

    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1); // set the video image quality

    // start the Video Capture Intent
    startActivityForResult(intent, CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE);
}
```

When the startActivityForResult() (/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) method is executed, see a modified camera application interface. After the user finishes taking a video (or cancels the op the user interface returns to your application, and you must intercept the onActivityResult() (/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) method to receive the the intent and continue your application execution. For information on how to receive the completed see the next section.

Receiving camera intent result

Once you have constructed and executed an image or video camera intent, your application must be configured to receive the result of the intent. This section shows you how to intercept the callback fr camera intent so your application can do further processing of the captured image or video.

In order to receive the result of an intent, you must override the onActivityResult() (/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) in the activity that sta intent. The following example demonstrates how to override onActivityResult() (/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) to capture the result o image camera intent (#intent-image) or video camera intent (#intent-video) examples shown in the previou sections.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Image captured and saved to fileUri specified in the Intent
            Toast.makeText(this, "Image saved to:\n" +
                     data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // User cancelled the image capture
        } else {
            // Image capture failed, advise user
        }
    }

    if (requestCode == CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Video captured and saved to fileUri specified in the Intent
            Toast.makeText(this, "Video saved to:\n" +
                     data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // User cancelled the video capture
        } else {
            // Video capture failed, advise user
        }
    }
}
```

Once your activity receives a successful result, the captured image or video is available in the specif
location for your application to access.

Building a Camera App

Some developers may require a camera user interface that is customized to the look of their applica
provides special features. Creating a customized camera activity requires more code than using an i
(#intents), but it can provide a more compelling experience for your users.

Note: The following guide is for the older, deprecated Camera (/reference/android/hardware/Camera.html) **AP
new or advanced camera applications, the newer** android.hardware.camera2
(/reference/android/hardware/camera2/package-summary.html) **API is recommended.**

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

- **Detect and Access Camera** - Create code to check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class that extends SurfaceView and implements th
SurfaceHolder interface. This class previews the live images from the camera.
- **Build a Preview Layout** - Once you have the camera preview class, create a view layout that incorpor
preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for your interface controls to start image or video cap
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, your application must properly release it for use by oth
applications.

Camera hardware is a shared resource that must be carefully managed so your application does not
with other applications that may also want to use it. The following sections discusses how to detect
hardware, how to request access to a camera, how to capture pictures or video and how to release th
camera when your application is done using it.

> **Caution:** Remember to release the Camera (/reference/android/hardware/Camera.html) object by calling th
> Camera.release() (/reference/android/hardware/Camera.html#release()) when your application is done usir
> your application does not properly release the camera, all subsequent attempts to access the cam
> including those by your own application, will fail and may cause your or other applications to be sh

Detecting camera hardware

If your application does not specifically require a camera using a manifest declaration, you should cl
see if a camera is available at runtime. To perform this check, use the PackageManager.hasSystemFea

(/reference/android/content/pm/PackageManager.html#hasSystemFeature(java.lang.String)) method, as shown in the code below:

```
/** 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 facing camera for video calls. Android 2.3 (API Level 9) and later allows you to check the number of available on a device using the `Camera.getNumberOfCameras()` (/reference/android/hardware/Camera.html#getNumberOfCameras()) method.

Accessing cameras

If you have determined that the device on which your application is running has a camera, you must to access it by getting an instance of `Camera` (/reference/android/hardware/Camera.html) (unless you are usi intent to access the camera (#intents)).

To access the primary camera, use the `Camera.open()` (/reference/android/hardware/Camera.html#open()) me and be sure to catch any exceptions, as shown in the code below:

```
/** 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
}
```

> **Caution:** Always check for exceptions when using `Camera.open()` (/reference/android/hardware/Camera.html#open()). Failing to check for exceptions if the camera is in use o not exist will cause your application to be shut down by the system.

On devices running Android 2.3 (API Level 9) or higher, you can access specific cameras using `Camera.open(int)` (/reference/android/hardware/Camera.html#open(int)). The example code above will acces first, back-facing camera on a device with more than one camera.

Checking camera features

Once you obtain access to a camera, you can get further information about its capabilities using the `Camera.getParameters()` (/reference/android/hardware/Camera.html#getParameters()) method and checking th returned `Camera.Parameters` (/reference/android/hardware/Camera.Parameters.html) object for supported capa When using API Level 9 or higher, use the `Camera.getCameraInfo()` (/reference/android/hardware/Camera.html#getCameraInfo(int, android.hardware.Camera.CameraInfo)) to determine if a is on the front or back of the device, and the orientation of the image.

Creating a preview class

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

The following example code demonstrates how to create a basic camera preview class that can be i in a View (/reference/android/view/View.html) layout. This class implements SurfaceHolder.Callback (/reference/android/view/SurfaceHolder.Callback.html) in order to capture the callback events for creating an destroying the view, which are needed for assigning the camera preview input.

```
/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
    private SurfaceHolder mHolder;
    private Camera mCamera;

    public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;

        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    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());
        }
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        // empty. Take care of releasing the Camera preview in your activity.
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // If your preview can change or rotate, take care of those events here.
        // Make sure to stop the preview before resizing or reformatting it.

        if (mHolder.getSurface() == null){
          // preview surface does not exist
          return;
        }

        // stop preview before making changes
        try {
            mCamera.stopPreview();
        } catch (Exception e){
          // ignore: tried to stop a non-existent preview
        }

        // set preview size and make any resize, rotate or
        // reformatting changes here

        // start preview with new settings
        try {
            mCamera.setPreviewDisplay(mHolder);
            mCamera.startPreview();

        } catch (Exception e){
            Log.d(TAG, "Error starting camera preview: " + e.getMessage());
        }
    }
}
```

If you want to set a specific size for your camera preview, set this in the `surfaceChanged()` method as in the comments above. When setting preview size, you *must use* values from `getSupportedPreviewS` (/reference/android/hardware/Camera.Parameters.html#getSupportedPreviewSizes()). *Do not* set arbitrary values in tl `setPreviewSize()` (/reference/android/hardware/Camera.Parameters.html#setPreviewSize(int, int)) method.

Placing preview in a layout

A camera preview class, such as the example shown in the previous section, must be placed in the la an activity along with other user interface controls for taking a picture or video. This section shows y to build a basic layout and activity for the preview.

The following layout code provides a very basic view that can be used to display a camera preview. I example, the `FrameLayout` (/reference/android/widget/FrameLayout.html) element is meant to be the contair the camera preview class. This layout type is used so that additional picture information or controls overlayed on the live camera preview images.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <FrameLayout
    android:id="@+id/camera_preview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    />

  <Button
    android:id="@+id/button_capture"
    android:text="Capture"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    />
</LinearLayout>
```

On most devices, the default orientation of the camera preview is landscape. This example layout sp horizontal (landscape) layout and the code below fixes the orientation of the application to landscap simplicity in rendering a camera preview, you should change your application's preview activity orier to landscape by adding the following to your manifest.

```
<activity android:name=".CameraActivity"
          android:label="@string/app_name"

          android:screenOrientation="landscape">
          <!-- configure this activity to use landscape orientation -->
          <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Note: A camera preview does not have to be in landscape mode. Starting in Android 2.2 (API Level can use the `setDisplayOrientation()` (/reference/android/hardware/Camera.html#setDisplayOrientation(int)) to set the rotation of the preview image. In order to change preview orientation as the user re-orien phone, within the `surfaceChanged()` (/reference/android/view/SurfaceHolder.Callback.html#surfaceChanged(android.view.SurfaceHolder, int, int, int)) r of your preview class, first stop the preview with `Camera.stopPreview()` (/reference/android/hardware/Camera.html#stopPreview()) change the orientation and then start the preview with `Camera.startPreview()` (/reference/android/hardware/Camera.html#startPreview()).

In the activity for your camera view, add your preview class to the `FrameLayout` (/reference/android/widget/FrameLayout.html) element shown in the example above. Your camera activity m ensure that it releases the camera when it is paused or shut down. The following example shows ho modify a camera activity to attach the preview class shown in Creating a preview class (#camera-previe

```
public class CameraActivity extends Activity {

    private Camera mCamera;
    private CameraPreview mPreview;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Create an instance of Camera
        mCamera = getCameraInstance();

        // Create our Preview view and set it as the content of our activity.
        mPreview = new CameraPreview(this, mCamera);
        FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
        preview.addView(mPreview);
    }
}
```

Note: The `getCameraInstance()` method in the example above refers to the example method show Accessing cameras (#access-camera).

Capturing pictures

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

In order to retrieve a picture, use the `Camera.takePicture()` (/reference/android/hardware/Camera.html#takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) method. This method takes parameters which receive data from the camera. In order to receive data in a JPEG format, you must implement an `Camera.PictureCallback` (/reference/android/hardware/Camera.PictureCallback.html) interface receive the image data and write it to a file. The following code shows a basic implementation of the `Camera.PictureCallback` (/reference/android/hardware/Camera.PictureCallback.html) interface to save an ima received from the camera.

```
private PictureCallback mPicture = new PictureCallback() {
    @Override
    public void onPictureTaken(byte[] data, Camera camera) {

        File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
        if (pictureFile == null){
            Log.d(TAG, "Error creating media file, check storage permissions: " +
                e.getMessage());
            return;
        }

        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        } catch (FileNotFoundException e) {
            Log.d(TAG, "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.d(TAG, "Error accessing file: " + e.getMessage());
        }
    }
};
```

Trigger capturing an image by calling the Camera.takePicture() (/reference/android/hardware/Camera.html#takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) method. The following exam code shows how to call this method from a button View.OnClickListener (/reference/android/view/View.OnClickListener.html).

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

Note: The `mPicture` member in the following example refers to the example code above.

Caution: Remember to release the Camera (/reference/android/hardware/Camera.html) object by calling th Camera.release() (/reference/android/hardware/Camera.html#release()) when your application is done usir information about how to release the camera, see Releasing the camera (#release-camera).

Capturing videos

Video capture using the Android framework requires careful management of the Camera (/reference/android/hardware/Camera.html) object and coordination with the MediaRecorder (/reference/android/media/MediaRecorder.html) class. When recording video with Camera (/reference/android/hardware/Camera.html), you must manage the Camera.lock() (/reference/android/hardware/Camera.html#lock()) and Camera.unlock() (/reference/android/hardware/Camera.html# calls to allow MediaRecorder (/reference/android/media/MediaRecorder.html) access to the camera hardware addition to the Camera.open() (/reference/android/hardware/Camera.html#open()) and Camera.release() (/reference/android/hardware/Camera.html#release()) calls.

Note: Starting with Android 4.0 (API level 14), the Camera.lock() (/reference/android/hardware/Camera.ht and Camera.unlock() (/reference/android/hardware/Camera.html#unlock()) calls are managed for you automatically.

Unlike taking pictures with a device camera, capturing video requires a very particular call order. You follow a specific order of execution to successfully prepare for and capture video with your applicati detailed below.

1. **Open Camera** - Use the Camera.open() to get an instance of the camera object.
2. **Connect Preview** - Prepare a live camera image preview by connecting a SurfaceView to the ca using Camera.setPreviewDisplay().
3. **Start Preview** - Call Camera.startPreview() to begin displaying the live camera images.
4. **Start Recording Video** - The following steps must be completed *in order* to successfully record
   a. **Unlock the Camera** - Unlock the camera for use by MediaRecorder by calling Camera.unlo
   b. **Configure MediaRecorder** - Call in the following MediaRecorder methods *in this order*. For information, see the MediaRecorder reference documentation.
      1. setCamera() - Set the camera to be used for video capture, use your application's c instance of Camera.
      2. setAudioSource() - Set the audio source, use MediaRecorder.AudioSource.CAMCORI
      3. setVideoSource() - Set the video source, use MediaRecorder.VideoSource.CAMERA.
      4. Set the video output format and encoding. For Android 2.2 (API Level 8) and highe the MediaRecorder.setProfile method, and get a profile instance using CamcorderProfile.get(). For versions of Android prior to 2.2, you must set the vid output format and encoding parameters:
         i. setOutputFormat() - Set the output format, specify the default setting or MediaRecorder.OutputFormat.MPEG_4.
         ii. setAudioEncoder() - Set the sound encoding type, specify the default settin MediaRecorder.AudioEncoder.AMR_NB.
         iii. setVideoEncoder() - Set the video encoding type, specify the default settinc MediaRecorder.VideoEncoder.MPEG_4_SP.
      5. setOutputFile() - Set the output file, use getOutputMediaFile(MEDIA_TYPE_VIDEO).toString() from the example method in Saving Media Files section.
      6. setPreviewDisplay() - Specify the SurfaceView preview layout element for your application. Use the same object you specified for **Connect Preview**.

      Caution: You must call these MediaRecorder (/reference/android/media/MediaRecorder.html) configuration methods *in this order*, otherwise your application will encounter errors a recording will fail.

   c. **Prepare MediaRecorder** - Prepare the MediaRecorder with provided configuration setting calling MediaRecorder.prepare().
   d. **Start MediaRecorder** - Start recording video by calling MediaRecorder.start().
5. **Stop Recording Video** - Call the following methods *in order*, to successfully complete a video re
   a. **Stop MediaRecorder** - Stop recording video by calling MediaRecorder.stop().
   b. **Reset MediaRecorder** - Optionally, remove the configuration settings from the recorder b calling MediaRecorder.reset().
   c. **Release MediaRecorder** - Release the MediaRecorder by calling MediaRecorder.release(
   d. **Lock the Camera** - Lock the camera so that future MediaRecorder sessions can use it by c Camera.lock(). Starting with Android 4.0 (API level 14), this call is not required unless th MediaRecorder.prepare() call fails.
6. **Stop the Preview** - When your activity has finished using the camera, stop the preview using Camera.stopPreview().
7. **Release Camera** - Release the camera so that other applications can use it by calling Camera.release().

Note: It is possible to use MediaRecorder (/reference/android/media/MediaRecorder.html) without creating camera preview first and skip the first few steps of this process. However, since users typically pre see a preview before starting a recording, that process is not discussed here.

Tip: If your application is typically used for recording video, set setRecordingHint(boolean) (/reference/android/hardware/Camera.Parameters.html#setRecordingHint(boolean)) to true prior to starting your p This setting can help reduce the time it takes to start recording.

Configuring MediaRecorder

When using the MediaRecorder (/reference/android/media/MediaRecorder.html) class to record video, you mu perform configuration steps in a *specific order* and then call the MediaRecorder.prepare() (/reference/android/media/MediaRecorder.html#prepare()) method to check and implement the configuration. following example code demonstrates how to properly configure and prepare the MediaRecorder (/reference/android/media/MediaRecorder.html) class for video recording.

```
private boolean prepareVideoRecorder(){

    mCamera = getCameraInstance();
    mMediaRecorder = new MediaRecorder();

    // Step 1: Unlock and set camera to MediaRecorder
    mCamera.unlock();
    mMediaRecorder.setCamera(mCamera);

    // Step 2: Set sources
    mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
    mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

    // Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
    mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));

    // Step 4: Set output file
    mMediaRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toString());

    // Step 5: Set the preview output
    mMediaRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());

    // Step 6: Prepare configured MediaRecorder
    try {
        mMediaRecorder.prepare();
    } catch (IllegalStateException e) {
        Log.d(TAG, "IllegalStateException preparing MediaRecorder: " + e.getMessage());
        releaseMediaRecorder();
        return false;
    } catch (IOException e) {
        Log.d(TAG, "IOException preparing MediaRecorder: " + e.getMessage());
        releaseMediaRecorder();
        return false;
    }
    return true;
}
```

Prior to Android 2.2 (API Level 8), you must set the output format and encoding formats parameters d instead of using CamcorderProfile (/reference/android/media/CamcorderProfile.html). This approach is demonstrated in the following code:

```
    // Step 3: Set output format and encoding (for versions prior to API Level 8)
    mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
    mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
    mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);
```

The following video recording parameters for MediaRecorder (/reference/android/media/MediaRecorder.html) given default settings, however, you may want to adjust these settings for your application:

- setVideoEncodingBitRate()
- setVideoSize()

- setVideoFrameRate()
- setAudioEncodingBitRate()
- setAudioChannels()
- setAudioSamplingRate()

Starting and stopping MediaRecorder

When starting and stopping video recording using the MediaRecorder (/reference/android/media/MediaReco
class, you must follow a specific order, as listed below.

1. Unlock the camera with Camera.unlock()
2. Configure MediaRecorder as shown in the code example above
3. Start recording using MediaRecorder.start()
4. Record the video
5. Stop recording using MediaRecorder.stop()
6. Release the media recorder with MediaRecorder.release()
7. Lock the camera using Camera.lock()

The following example code demonstrates how to wire up a button to properly start and stop video re
using the camera and the MediaRecorder (/reference/android/media/MediaRecorder.html) class.

> **Note:** When completing a video recording, do not release the camera or else your preview will be s

```
private boolean isRecording = false;

// Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
    new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (isRecording) {
                // stop recording and release camera
                mMediaRecorder.stop();  // stop the recording
                releaseMediaRecorder(); // release the MediaRecorder object
                mCamera.lock();         // take camera access back from MediaRecorder

                // inform the user that recording has stopped
                setCaptureButtonText("Capture");
                isRecording = false;
            } else {
                // initialize video camera
                if (prepareVideoRecorder()) {
                    // Camera is available and unlocked, MediaRecorder is prepared,
                    // now you can start recording
                    mMediaRecorder.start();

                    // inform the user that recording has started
                    setCaptureButtonText("Stop");
                    isRecording = true;
                } else {
                    // prepare didn't work, release the camera
                    releaseMediaRecorder();
                    // inform user
                }
            }
        }
    }
);
```

Note: In the above example, the `prepareVideoRecorder()` method refers to the example code show Configuring MediaRecorder (#configuring-mediarecorder). This method takes care of locking the camera configuring and preparing the `MediaRecorder` (/reference/android/media/MediaRecorder.html) instance.

Releasing the camera

Cameras are a resource that is shared by applications on a device. Your application can make use of camera after getting an instance of `Camera` (/reference/android/hardware/Camera.html), and you must be par careful to release the camera object when your application stops using it, and as soon as your applic paused (`Activity.onPause()` (/reference/android/app/Activity.html#onPause())). If your application does not properly release the camera, all subsequent attempts to access the camera, including those by your application, will fail and may cause your or other applications to be shut down.

To release an instance of the `Camera` (/reference/android/hardware/Camera.html) object, use the `Camera.rel` (/reference/android/hardware/Camera.html#release()) method, as shown in the example code below.

```
public class CameraActivity extends Activity {
    private Camera mCamera;
    private SurfaceView mPreview;
    private MediaRecorder mMediaRecorder;

    ...

    @Override
    protected void onPause() {
        super.onPause();
        releaseMediaRecorder();       // if you are using MediaRecorder, release it fir
        releaseCamera();              // release the camera immediately on pause event
    }

    private void releaseMediaRecorder(){
        if (mMediaRecorder != null) {
            mMediaRecorder.reset();   // clear recorder configuration
            mMediaRecorder.release(); // release the recorder object
            mMediaRecorder = null;
            mCamera.lock();           // lock camera for later use
        }
    }

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

Caution: If your application does not properly release the camera, all subsequent attempts to acce camera, including those by your own application, will fail and may cause your or other applications shut down.

Saving Media Files

Media files created by users such as pictures and videos should be saved to a device's external stora directory (SD Card) to conserve system space and to allow users to access these files without their c There are many possible directory locations to save media files on a device, however there are only t standard locations you should consider as a developer:

- `Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)` - This method

the standard, shared and recommended location for saving pictures and videos. This directory is sha
(public), so other applications can easily discover, read, change and delete files saved in this locatio
application is uninstalled by the user, media files saved to this location will not be removed. To avoi
interfering with users existing pictures and videos, you should create a sub-directory for your applica
media files within this directory, as shown in the code sample below. This method is available in An
(API Level 8), for equivalent calls in earlier API versions, see Saving Shared Files.

- Context.getExternalFilesDir(Environment.DIRECTORY_PICTURES) - This method returns a standard lo
for saving pictures and videos which are associated with your application. If your application is unin
any files saved in this location are removed. Security is not enforced for files in this location and oth
applications may read, change and delete them.

The following example code demonstrates how to create a File (/reference/java/io/File.html) or Uri
(/reference/android/net/Uri.html) location for a media file that can be used when invoking a device's cam
an Intent (/reference/android/content/Intent.html) or as part of a Building a Camera App (#custom-camera).

```
public static final int MEDIA_TYPE_IMAGE = 1;
public static final int MEDIA_TYPE_VIDEO = 2;

/** Create a file Uri for saving an image or video */
private static Uri getOutputMediaFileUri(int type){
      return Uri.fromFile(getOutputMediaFile(type));
}

/** 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;
        }
    }

    // Create a media file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    File mediaFile;
    if (type == MEDIA_TYPE_IMAGE){
        mediaFile = new File(mediaStorageDir.getPath() + File.separator +
        "IMG_"+ timeStamp + ".jpg");
    } else if(type == MEDIA_TYPE_VIDEO) {
        mediaFile = new File(mediaStorageDir.getPath() + File.separator +
        "VID_"+ timeStamp + ".mp4");
    } else {
        return null;
    }

    return mediaFile;
}
```

Note: Environment.getExternalStoragePublicDirectory()
(/reference/android/os/Environment.html#getExternalStoragePublicDirectory(java.lang.String)) is available in An
(API Level 8) or higher. If you are targeting devices with earlier versions of Android, use

> Environment.getExternalStorageDirectory() (/reference/android/os/Environment.html#getExternalStorageDirectory()) instead. For more information, see S Shared Files (/guide/topics/data/data-storage.html#SavingSharedFiles).

For more information about saving files on an Android device, see Data Storage (/guide/topics/data/data-storage.html).

Camera Features

Android supports a wide array of camera features you can control with your camera application, sucl picture format, flash mode, focus settings, and many more. This section lists the common camera fe and briefly discusses how to use them. Most camera features can be accessed and set using the thr `Camera.Parameters` (/reference/android/hardware/Camera.Parameters.html) object. However, there are several important features that require more than simple settings in `Camera.Parameters` (/reference/android/hardware/Camera.Parameters.html). These features are covered in the following sections:

- Metering and focus areas
- Face detection
- Time lapse video

For general information about how to use features that are controlled through `Camera.Parameters` (/reference/android/hardware/Camera.Parameters.html), review the Using camera features (#using-features) sectio more detailed information about how to use features controlled through the camera parameters obje follow the links in the feature list below to the API reference documentation.

Table 1. Common camera features sorted by the Android API Level in which they were introduced.

Feature	API Level	Description
Face Detection	14	Identify human faces within a picture and use them for focus, metering and balance
Metering Areas	14	Specify one or more areas within an image for calculating white balance
Focus Areas	14	Set one or more areas within an image to use for focus
`White Balance Lock`	14	Stop or start automatic white balance adjustments
`Exposure Lock`	14	Stop or start automatic exposure adjustments
`Video Snapshot`	14	Take a picture while shooting video (frame grab)
Time Lapse Video	11	Record frames with set delays to record a time lapse video
`Multiple Cameras`	9	Support for more than one camera on a device, including front-facing and b facing cameras
`Focus Distance`	9	Reports distances between the camera and objects that appear to be in foc
`Zoom`	8	Set image magnification
`Exposure Compensation`	8	Increase or decrease the light exposure level
`GPS Data`	5	Include or omit geographic location data with the image
`White Balance`	5	Set the white balance mode, which affects color values in the captured ima
`Focus Mode`	5	Set how the camera focuses on a subject such as automatic, fixed, macro c infinity
`Scene Mode`	5	Apply a preset mode for specific types of photography situations such as ni beach, snow or candlelight scenes
`JPEG Quality`	5	Set the compression level for a JPEG image, which increases or decreases output file quality and size
`Flash Mode`	5	Turn flash on, off, or use automatic setting
`Color Effects`	5	Apply a color effect to the captured image such as black and white, sepia t negative.
`Anti-Banding`	5	Reduces the effect of banding in color gradients due to JPEG compression

`Picture Format`	1	Specify the file format for the picture
`Picture Size`	1	Specify the pixel dimensions of the saved picture

> **Note:** These features are not supported on all devices due to hardware differences and software implementation. For information on checking the availability of features on the device where your application is running, see Checking feature availability (#check-feature).

Checking feature availability

The first thing to understand when setting out to use camera features on Android devices is that not a camera features are supported on all devices. In addition, devices that support a particular feature m support them to different levels or with different options. Therefore, part of your decision process as develop a camera application is to decide what camera features you want to support and to what lev making that decision, you should plan on including code in your camera application that checks to se device hardware supports those features and fails gracefully if a feature is not available.

You can check the availabilty of camera features by getting an instance of a camera's parameters ob and checking the relevant methods. The following code sample shows you how to obtain a `Camera.Parameters (/reference/android/hardware/Camera.Parameters.html)` object and check if the camera sup the autofocus feature:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();

List<String> focusModes = params.getSupportedFocusModes();
if (focusModes.contains(Camera.Parameters.FOCUS_MODE_AUTO)) {
  // Autofocus mode is supported
}
```

You can use the technique shown above for most camera features. The `Camera.Parameters (/reference/android/hardware/Camera.Parameters.html)` object provides a `getSupported...()`, `is...Supported(` `getMax...()` method to determine if (and to what extent) a feature is supported.

If your application requires certain camera features in order to function properly, you can require the through additions to your application manifest. When you declare the use of specific camera feature as flash and auto-focus, Google Play restricts your application from being installed on devices which support these features. For a list of camera features that can be declared in your app manifest, see th manifest Features Reference (/guide/topics/manifest/uses-feature-element.html#hw-features).

Using camera features

Most camera features are activated and controlled using a `Camera.Parameters (/reference/android/hardware/Camera.Parameters.html)` object. You obtain this object by first getting an instanc `Camera (/reference/android/hardware/Camera.html)` object, calling the `getParameters() (/reference/android/hardware/Camera.html#getParameters())` method, changing the returned parameter object a setting it back into the camera object, as demonstrated in the following example code:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();
// set the focus mode
params.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
// set Camera parameters
mCamera.setParameters(params);
```

This technique works for nearly all camera features, and most parameters can be changed at any tim you have obtained an instance of the `Camera (/reference/android/hardware/Camera.html)` object. Changes to parameters are typically visible to the user immediately in the application's camera preview. On the side, parameter changes may take several frames to actually take effect as the camera hardware pro the new instructions and then sends updated image data.

Important: Some camera features cannot be changed at will. In particular, changing the size or ori
of the camera preview requires that you first stop the preview, change the preview size, and then re
the preview. Starting with Android 4.0 (API Level 14) preview orientation can be changed without r
the preview.

Other camera features require more code in order to implement, including:

- Metering and focus areas
- Face detection
- Time lapse video

A quick outline of how to implement these features is provided in the following sections.

Metering and focus areas

In some photographic scenarios, automatic focusing and light metering may not produce the desired
Starting with Android 4.0 (API Level 14), your camera application can provide additional controls to a
your app or users to specify areas in an image to use for determining focus or light level settings and
these values to the camera hardware for use in capturing images or video.

Areas for metering and focus work very similarly to other camera features, in that you control them t
methods in the Camera.Parameters (/reference/android/hardware/Camera.Parameters.html) object. The followi
demonstrates setting two light metering areas for an instance of Camera (/reference/android/hardware/Cam

```
// Create an instance of Camera
mCamera = getCameraInstance();

// set Camera parameters
Camera.Parameters params = mCamera.getParameters();

if (params.getMaxNumMeteringAreas() > 0){ // check that metering areas are supported
    List<Camera.Area> meteringAreas = new ArrayList<Camera.Area>();

    Rect areaRect1 = new Rect(-100, -100, 100, 100);    // specify an area in center of
    meteringAreas.add(new Camera.Area(areaRect1, 600)); // set weight to 60%
    Rect areaRect2 = new Rect(800, -1000, 1000, -800);  // specify an area in upper rig
    meteringAreas.add(new Camera.Area(areaRect2, 400)); // set weight to 40%
    params.setMeteringAreas(meteringAreas);
}

mCamera.setParameters(params);
```

The Camera.Area (/reference/android/hardware/Camera.Area.html) object contains two data parameters: A Re
(/reference/android/graphics/Rect.html) object for specifying an area within the camera's field of view and
weight value, which tells the camera what level of importance this area should be given in light mete
focus calculations.

The Rect (/reference/android/graphics/Rect.html) field in a Camera.Area (/reference/android/hardware/Camera.Are
object describes a rectangular shape mapped on a 2000 x 2000 unit grid. The coordinates -1000, -100
represent the top, left corner of the camera image, and coordinates 1000, 1000 represent the bottom,
corner of the camera image, as shown in the illustration below.

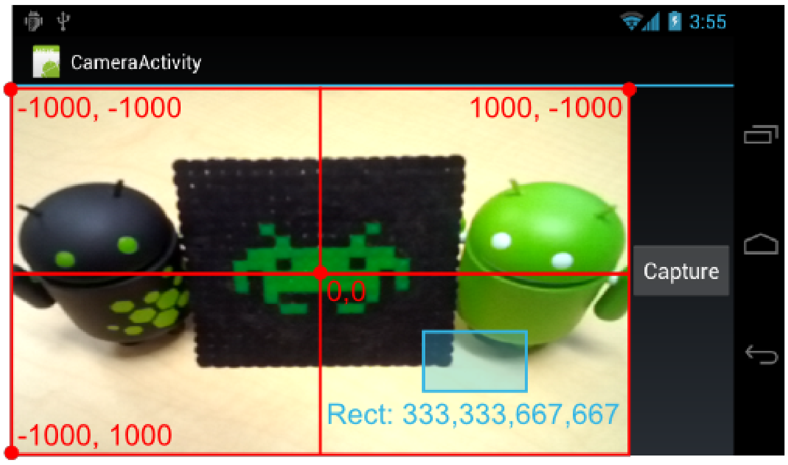

Figure 1. The red lines illustrate the coordinate system for specifying a Camera.Area (/reference/android/hardware/Camera.Area.html) within a camera preview. The blue box shows the location and shap camera area with the Rect (/reference/android/graphics/Rect.html) values 333,333,667,667.

The bounds of this coordinate system always correspond to the outer edge of the image visible in the preview and do not shrink or expand with the zoom level. Similarly, rotation of the image preview usi Camera.setDisplayOrientation() (/reference/android/hardware/Camera.html#setDisplayOrientation(int)) does r remap the coordinate system.

Face detection

For pictures that include people, faces are usually the most important part of the picture, and should for determining both focus and white balance when capturing an image. The Android 4.0 (API Level framework provides APIs for identifying faces and calculating picture settings using face recognitio technology.

> **Note:** While the face detection feature is running, setWhiteBalance(String) (/reference/android/hardware/Camera.Parameters.html#setWhiteBalance(java.lang.String)), setFocusAreas(List) (/reference/android/hardware/Camera.Parameters.html#setFocusAreas(java.util.List<android.hardware.Camera.Area>)) setMeteringAreas(List) (/reference/android/hardware/Camera.Parameters.html#setMeteringAreas(java.util.List<android.hardware.Camera.Area no effect.

Using the face detection feature in your camera application requires a few general steps:

- Check that face detection is supported on the device
- Create a face detection listener
- Add the face detection listener to your camera object
- Start face detection after preview (and after *every* preview restart)

The face detection feature is not supported on all devices. You can check that this feature is support calling getMaxNumDetectedFaces() (/reference/android/hardware/Camera.Parameters.html#getMaxNumDetectedFaces(example of this check is shown in the startFaceDetection() sample method below.

In order to be notified and respond to the detection of a face, your camera application must set a list face detection events. In order to do this, you must create a listener class that implements the Camera.FaceDetectionListener (/reference/android/hardware/Camera.FaceDetectionListener.html) interface as in the example code below.

```
class MyFaceDetectionListener implements Camera.FaceDetectionListener {

    @Override
    public void onFaceDetection(Face[] faces, Camera camera) {
        if (faces.length > 0){
            Log.d("FaceDetection", "face detected: "+ faces.length +
                    " Face 1 Location X: " + faces[0].rect.centerX() +
                    "Y: " + faces[0].rect.centerY() );
            }
        }
    }
```

After creating this class, you then set it into your application's Camera (/reference/android/hardware/Camera object, as shown in the example code below:

```
mCamera.setFaceDetectionListener(new MyFaceDetectionListener());
```

Your application must start the face detection function each time you start (or restart) the camera pr Create a method for starting face detection so you can call it as needed, as shown in the example co 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();
    }
}
```

You must start face detection *each time* you start (or restart) the camera preview. If you use the previ shown in Creating a preview class (#camera-preview), add your startFaceDetection() (/reference/android/hardware/Camera.html#startFaceDetection()) method to both the surfaceCreated() (/reference/android/view/SurfaceHolder.Callback.html#surfaceCreated(android.view.SurfaceHolder)) and surfaceChar (/reference/android/view/SurfaceHolder.Callback.html#surfaceChanged(android.view.SurfaceHolder, int, int, int)) me your preview class, as shown in the sample code below.

```
public void surfaceCreated(SurfaceHolder holder) {
    try {
        mCamera.setPreviewDisplay(holder);
        mCamera.startPreview();

        startFaceDetection(); // start face detection feature

    } catch (IOException e) {
        Log.d(TAG, "Error setting camera preview: " + e.getMessage());
    }
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {

    if (mHolder.getSurface() == null){
        // preview surface does not exist
        Log.d(TAG, "mHolder.getSurface() == null");
        return;
    }

    try {
        mCamera.stopPreview();

    } catch (Exception e){
        // ignore: tried to stop a non-existent preview
        Log.d(TAG, "Error stopping camera preview: " + e.getMessage());
    }

    try {
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();

        startFaceDetection(); // re-start face detection feature

    } catch (Exception e){
        // ignore: tried to stop a non-existent preview
        Log.d(TAG, "Error starting camera preview: " + e.getMessage());
    }
}
```

Note: Remember to call this method *after* calling `startPreview()` (/reference/android/hardware/Camera.html#startPreview()). Do not attempt to start face detection in the `onCr` (/reference/android/app/Activity.html#onCreate(android.os.Bundle)) method of your camera app's main activ the preview is not available by this point in your application's the execution.

Time lapse video

Time lapse video allows users to create video clips that combine pictures taken a few seconds or mi apart. This feature uses `MediaRecorder` (/reference/android/media/MediaRecorder.html) to record the images time lapse sequence.

To record a time lapse video with `MediaRecorder` (/reference/android/media/MediaRecorder.html), you must c the recorder object as if you are recording a normal video, setting the captured frames per second to number and using one of the time lapse quality settings, as shown in the code example below.

```
// Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_TIME_LAPSE_HIGH
...
// Step 5.5: Set the video capture rate to a low number
mMediaRecorder.setCaptureRate(0.1); // capture a frame every 10 seconds
```

These settings must be done as part of a larger configuration procedure for `MediaRecorder` (/reference/android/media/MediaRecorder.html). For a full configuration code example, see Configuring MediaRecorder (#configuring-mediarecorder). Once the configuration is complete, you start the video reco if you were recording a normal video clip. For more information about configuring and running `MediaR` (/reference/android/media/MediaRecorder.html), see Capturing videos (#capture-video).