public class

Added in API level 1
Deprecated since API level 21

Camera

extends Object

java.lang.Object
↳android.hardware.Camera

This class was deprecated in API level 21.
We recommend using the new android.hardware.camera2 (/reference/android/hardware/camera2/package-summary.html) API for new applications.

Class Overview

The Camera class is used to set image capture settings, start/stop preview, snap pictures, and retrieve frames for encoding for video. This class is a client for the Camera service, which manages the actual camera hardware.

To access the device camera, you must declare the CAMERA (/reference/android/Manifest.permission.html#CAMERA) permission in your Android Manifest. Also be sure to include the <uses-feature> (/guide/topics/manifest/uses-feature-element.html) manifest element to declare camera features used by your application. For example, if you use the camera and auto-focus feature, your Manifest should include the following:

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

To take pictures with this class, use the following steps:

1. Obtain an instance of Camera from `open(int)`.
2. Get existing (default) settings with `getParameters()`.
3. If necessary, modify the returned `Camera.Parameters` object and call `setParameters(Camera.Parameters)`.
4. If desired, call `setDisplayOrientation(int)`.
5. **Important**: Pass a fully initialized `SurfaceHolder` to `setPreviewDisplay(SurfaceHolder)`. Without a surface, the camera will be unable to start the preview.
6. **Important**: Call `startPreview()` to start updating the preview surface. Preview must be started before you can take a picture.
7. When you want, call `takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback, Camera.PictureCallback)` to capture a photo. Wait for the callbacks to provide the actual image data.
8. After taking a picture, preview display will have stopped. To take more photos, call `startPreview()` again first.
9. Call `stopPreview()` to stop updating the preview surface.
10. **Important:** Call `release()` to release the camera for use by other applications. Applications should release the camera immediately in `onPause()` (and re-`open()` it in `onResume()`).

To quickly switch to video recording mode, use these steps:

1. Obtain and initialize a Camera and start preview as described above.
2. Call `unlock()` to allow the media process to access the camera.
3. Pass the camera to `setCamera(Camera)`. See `MediaRecorder` information about video recording.
4. When finished recording, call `reconnect()` to re-acquire and re-lock the camera.
5. If desired, restart preview and take more photos or videos.
6. Call `stopPreview()` and `release()` as described above.

This class is not thread-safe, and is meant for use from one event thread. Most long-running operations (preview, focus, photo capture, etc) happen asynchronously and invoke callbacks as necessary. Callbacks will be invoked on the event thread `open(int) (/reference/android/hardware/Camera.html#open(int))` was called from. This class's methods must never be called from multiple threads at once.

> **Caution:** Different Android-powered devices may have different hardware specifications, such as megapixel ratings and auto-focus capabilities. In order for your application to be compatible with more devices, you should not make assumptions about the device camera specifications.

Developer Guides

For more information about using cameras, read the Camera (/guide/topics/media/camera.html) developer guide.

Summary

Nested Classes	
class Camera.Area	*This class was deprecated in API level 21. We recommend using the new `android.hardware.camera2` API for new applications.*
interface Camera.AutoFocusCallback	*This interface was deprecated in API level 21. We recommend using the new `android.hardware.camera2` API for new applications.*
interface Camera.AutoFocusMoveCallback	*This interface was deprecated in API level 21. We recommend using the new `android.hardware.camera2` API for new applications.*
class Camera.CameraInfo	*This class was deprecated in API level 21. We recommend using the new `android.hardware.camera2` API for new applications.*
interface Camera.ErrorCallback	*This interface was deprecated in API level 21. We recommend using the new `android.hardware.camera2` API for new applications.*
class Camera.Face	*This class was deprecated in API level 21. We recommend using the new `android.hardware.camera2` API for new applications.*
	This interface was deprecated in API level 21. We recommend using

interface Camera.FaceDetectionListener	*the new* `android.hardware.camera2` *API for new applications.*
interface Camera.OnZoomChangeListener	*This interface was deprecated in API level 21. We recommend using the new* `android.hardware.camera2` *API for new applications.*
class Camera.Parameters	*This class was deprecated in API level 21. We recommend using the new* `android.hardware.camera2` *API for new applications.*
interface Camera.PictureCallback	*This interface was deprecated in API level 21. We recommend using the new* `android.hardware.camera2` *API for new applications.*
interface Camera.PreviewCallback	*This interface was deprecated in API level 21. We recommend using the new* `android.hardware.camera2` *API for new applications.*
interface Camera.ShutterCallback	*This interface was deprecated in API level 21. We recommend using the new* `android.hardware.camera2` *API for new applications.*
class Camera.Size	*This class was deprecated in API level 21. We recommend using the new* `android.hardware.camera2` *API for new applications.*

Constants

String ACTION_NEW_PICTURE	Broadcast Action: A new picture is taken by the camera, and the entry of the picture has been added to the media store.
String ACTION_NEW_VIDEO	Broadcast Action: A new video is recorded by the camera, and the entry of the video has been added to the media store.
int CAMERA_ERROR_SERVER_DIED	Media server died.
int CAMERA_ERROR_UNKNOWN	Unspecified camera error.

Public Methods

final void	addCallbackBuffer (byte[] callbackBuffer) Adds a pre-allocated buffer to the preview callback buffer queue.
final void	autoFocus (Camera.AutoFocusCallback cb) Starts camera auto-focus and registers a callback function to run when the c
final void	cancelAutoFocus () Cancels any auto-focus function in progress.
final boolean	enableShutterSound (boolean enabled) Enable or disable the default shutter sound when taking a picture.
static void	getCameraInfo (int cameraId, Camera.CameraInfo cameraInfo) Returns the information about a particular camera.
static int	getNumberOfCameras () Returns the number of physical cameras available on this device.
	getParameters ()

Camera.Parameters	Returns the current settings for this Camera service.
final void	lock () Re-locks the camera to prevent other processes from accessing it.
static Camera	open (int cameraId) Creates a new Camera object to access a particular hardware camera.
static Camera	open () Creates a new Camera object to access the first back-facing camera on the
final void	reconnect () Reconnects to the camera service after another process used it.
final void	release () Disconnects and releases the Camera object resources.
void	setAutoFocusMoveCallback (Camera.AutoFocusMoveCallback cb) Sets camera auto-focus move callback.
final void	setDisplayOrientation (int degrees) Set the clockwise rotation of preview display in degrees.
final void	setErrorCallback (Camera.ErrorCallback cb) Registers a callback to be invoked when an error occurs.
final void	setFaceDetectionListener (Camera.FaceDetectionListener listener) Registers a listener to be notified about the faces detected in the preview fra
final void	setOneShotPreviewCallback (Camera.PreviewCallback cb) Installs a callback to be invoked for the next preview frame in addition to dis
void	setParameters (Camera.Parameters params) Changes the settings for this Camera service.
final void	setPreviewCallback (Camera.PreviewCallback cb) Installs a callback to be invoked for every preview frame in addition to displa
final void	setPreviewCallbackWithBuffer (Camera.PreviewCallback cb) Installs a callback to be invoked for every preview frame, using buffers suppl `(/reference/android/hardware/Camera.html#addCallbackBuffer(byte[]))`, in addition to dis
final void	setPreviewDisplay (SurfaceHolder holder) Sets the `Surface` to be used for live preview.
final void	setPreviewTexture (SurfaceTexture surfaceTexture) Sets the `SurfaceTexture` to be used for live preview.
final void	setZoomChangeListener (Camera.OnZoomChangeListener listener) Registers a listener to be notified when the zoom value is updated by the car
final void	startFaceDetection () Starts the face detection.
final void	startPreview () Starts capturing and drawing preview frames to the screen.
final void	startSmoothZoom (int value) Zooms to the requested value smoothly.
final void	stopFaceDetection () Stops the face detection.
final void	stopPreview () Stops capturing and drawing preview frames to the surface, and resets the c
final void	stopSmoothZoom () Stops the smooth zoom.
final void	takePicture (Camera.ShutterCallback shutter, Camera.PictureCallback raw, Ca Equivalent to takePicture(shutter, raw, null, jpeg).
final void	takePicture (Camera.ShutterCallback shutter, Camera.PictureCallback raw, Ca

	Triggers an asynchronous image capture.
final void	unlock() Unlocks the camera to allow another process to access it.

Protected Methods

void	finalize() Invoked when the garbage collector has detected that this instance is no longer reachable.

Inherited Methods [Expand]

▸ From class java.lang.Object

Constants

public static final String **ACTION_NEW_PICTURE** Added in API level 14

Broadcast Action: A new picture is taken by the camera, and the entry of the picture has been added to the media store. `getData()` (/reference/android/content/Intent.html#getData()) is URI of the picture.

Constant Value: "android.hardware.action.NEW_PICTURE"

public static final String **ACTION_NEW_VIDEO** Added in API level 14

Broadcast Action: A new video is recorded by the camera, and the entry of the video has been added to the media store. `getData()` (/reference/android/content/Intent.html#getData()) is URI of the video.

Constant Value: "android.hardware.action.NEW_VIDEO"

public static final int **CAMERA_ERROR_SERVER_DIED** Added in API level 1

Media server died. In this case, the application must release the Camera object and instantiate a new one.

See Also

`Camera.ErrorCallback`

Constant Value: 100 (0x00000064)

public static final int **CAMERA_ERROR_UNKNOWN** Added in API level 1

Unspecified camera error.

See Also

`Camera.ErrorCallback`

Constant Value: 1 (0x00000001)

Public Methods

public final void **addCallbackBuffer** (byte[] callbackBuffer) Added in API level 8

Adds a pre-allocated buffer to the preview callback buffer queue. Applications can add one or more buffers to the queue. When a preview frame arrives and there is still at least one available buffer, the buffer will be used and removed from the queue. Then preview callback is invoked with the buffer. If a frame arrives and there is no buffer left, the frame is discarded. Applications should add buffers back when they finish processing the data in them.

For formats besides YV12, the size of the buffer is determined by multiplying the preview image width, height, and bytes per pixel. The width and height can be read from `getPreviewSize()` (/reference/android/hardware/Camera.Parameters.html#getPreviewSize()). Bytes per pixel can be computed from `getBitsPerPixel(int)` (/reference/android/graphics/ImageFormat.html#getBitsPerPixel(int)) / 8, using the image format from `getPreviewFormat()` (/reference/android/hardware/Camera.Parameters.html#getPreviewFormat()).

If using the `YV12` (/reference/android/graphics/ImageFormat.html#YV12) format, the size can be calculated using the equations listed in `setPreviewFormat(int)` (/reference/android/hardware/Camera.Parameters.html#setPreviewFormat(int)).

This method is only necessary when `setPreviewCallbackWithBuffer(PreviewCallback)` (/reference/android/hardware/Camera.html#setPreviewCallbackWithBuffer(android.hardware.Camera.PreviewCallback)) is used. When `setPreviewCallback(PreviewCallback)` (/reference/android/hardware/Camera.html#setPreviewCallback(android.hardware.Camera.PreviewCallback)) or `setOneShotPreviewCallback(PreviewCallback)` (/reference/android/hardware/Camera.html#setOneShotPreviewCallback(android.hardware.Camera.PreviewCallback)) are used, buffers are automatically allocated. When a supplied buffer is too small to hold the preview frame data, preview callback will return null and the buffer will be removed from the buffer queue.

Parameters

callbackBuffer	the buffer to add to the queue. The size of the buffer must match the values described above.

See Also

`setPreviewCallbackWithBuffer(PreviewCallback)`

public final void **autoFocus** (Camera.AutoFocusCallback cb)

Added in API level 1

Starts camera auto-focus and registers a callback function to run when the camera is focused. This method is only valid when preview is active (between `startPreview()` (/reference/android/hardware/Camera.html#startPreview()) and before `stopPreview()` (/reference/android/hardware/Camera.html#stopPreview())).

Callers should check `getFocusMode()` (/reference/android/hardware/Camera.Parameters.html#getFocusMode()) to determine if this method should be called. If the camera does not support auto-focus, it is a no-op and `onAutoFocus(boolean, Camera)` (/reference/android/hardware/Camera.AutoFocusCallback.html#onAutoFocus(boolean, android.hardware.Camera)) callback will be called immediately.

If your application should not be installed on devices without auto-focus, you must declare that your application uses auto-focus with the <uses-feature> (/guide/topics/manifest/uses-feature-element.html) manifest element.

If the current flash mode is not `FLASH_MODE_OFF` (/reference/android/hardware/Camera.Parameters.html#FLASH_MODE_OFF), flash may be fired during auto-focus, depending on the driver and camera hardware.

Auto-exposure lock `getAutoExposureLock()`

(/reference/android/hardware/Camera.Parameters.html#getAutoExposureLock()) and auto-white balance locks getAutoWhiteBalanceLock() (/reference/android/hardware/Camera.Parameters.html#getAutoWhiteBalanceLock()) do not change during and after autofocus. But auto-focus routine may stop auto-exposure and auto-white balance transiently during focusing.

Stopping preview with stopPreview() (/reference/android/hardware/Camera.html#stopPreview()), or triggering still image capture with takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback) (/reference/android/hardware/Camera.html#takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)), will not change the the focus position. Applications must call cancelAutoFocus to reset the focus.

If autofocus is successful, consider using MediaActionSound (/reference/android/media/MediaActionSound.html) to properly play back an autofocus success sound to the user.

Parameters

cb	the callback to run

See Also

cancelAutoFocus()
setAutoExposureLock(boolean)
setAutoWhiteBalanceLock(boolean)
MediaActionSound

public final void **cancelAutoFocus** ()

Added in API level 5

Cancels any auto-focus function in progress. Whether or not auto-focus is currently in progress, this function will return the focus position to the default. If the camera does not support auto-focus, this is a no-op.

See Also

autoFocus(Camera.AutoFocusCallback)

public final boolean **enableShutterSound** (boolean enabled)

Added in API level 17

Enable or disable the default shutter sound when taking a picture.

By default, the camera plays the system-defined camera shutter sound when takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback) (/reference/android/hardware/Camera.html#takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) is called. Using this method, the shutter sound can be disabled. It is strongly recommended that an alternative shutter sound is played in the Camera.ShutterCallback (/reference/android/hardware/Camera.ShutterCallback.html) when the system shutter sound is disabled.

Note that devices may not always allow disabling the camera shutter sound. If the shutter sound state cannot be set to the desired value, this method will return false. canDisableShutterSound (/reference/android/hardware/Camera.CameraInfo.html#canDisableShutterSound) can be used to determine whether the device will allow the shutter sound to be disabled.

Parameters

enabled	whether the camera should play the system shutter sound when takePicture is called.

Returns

`true` if the shutter sound state was successfully changed. `false` if the shutter sound state could not be changed. `true` is also returned if shutter sound playback is already set to the requested state.

See Also

`takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback)`
`canDisableShutterSound`
`Camera.ShutterCallback`

public static void **getCameraInfo** (int cameraId, Camera.CameraInfo cameraInfo)

Added in API level 9

Returns the information about a particular camera. If `getNumberOfCameras()` (/reference/android/hardware/Camera.html#getNumberOfCameras()) returns N, the valid id is 0 to N-1.

public static int **getNumberOfCameras** ()

Added in API level 9

Returns the number of physical cameras available on this device.

public Camera.Parameters **getParameters** ()

Added in API level 1

Returns the current settings for this Camera service. If modifications are made to the returned Parameters, they must be passed to `setParameters(Camera.Parameters)` (/reference/android/hardware/Camera.html#setParameters(android.hardware.Camera.Parameters)) to take effect.

See Also

`setParameters(Camera.Parameters)`

public final void **lock** ()

Added in API level 5

Re-locks the camera to prevent other processes from accessing it. Camera objects are locked by default unless `unlock()` (/reference/android/hardware/Camera.html#unlock()) is called. Normally `reconnect()` (/reference/android/hardware/Camera.html#reconnect()) is used instead.

Since API level 14, camera is automatically locked for applications in `start()` (/reference/android/media/MediaRecorder.html#start()). Applications can use the camera (ex: zoom) after recording starts. There is no need to call this after recording starts or stops.

If you are not recording video, you probably do not need this method.

Throws

RuntimeException	if the camera cannot be re-locked (for example, if the camera is still in use by another process).

public static Camera **open** (int cameraId)

Added in API level 9

Creates a new Camera object to access a particular hardware camera. If the same camera is opened by other applications, this will throw a RuntimeException.

You must call `release()` (/reference/android/hardware/Camera.html#release()) when you are done using the camera, otherwise it will remain locked and be unavailable to other applications.

Your application should only have one Camera object active at a time for a particular hardware camera.

Callbacks from other methods are delivered to the event loop of the thread which called open(). If this thread has no event loop, then callbacks are delivered to the main application event loop. If there is no main application event loop, callbacks are not delivered.

> **Caution:** On some devices, this method may take a long time to complete. It is best to call this method from a worker thread (possibly using `AsyncTask` (/reference/android/os/AsyncTask.html)) to avoid blocking the main application UI thread.

Parameters

cameraId	the hardware camera to access, between 0 and `getNumberOfCameras()`-1.

Returns

a new Camera object, connected, locked and ready for use.

Throws

RuntimeException	if opening the camera fails (for example, if the camera is in use by another process or device policy manager has disabled the camera).

See Also

`getCameraDisabled(android.content.ComponentName)`

public static Camera **open** ()

Added in API level 1

Creates a new Camera object to access the first back-facing camera on the device. If the device does not have a back-facing camera, this returns null.

See Also

`open(int)`

public final void **reconnect** ()

Added in API level 8

Reconnects to the camera service after another process used it. After `unlock()` (/reference/android/hardware/Camera.html#unlock()) is called, another process may use the camera; when the process is done, you must reconnect to the camera, which will re-acquire the lock and allow you to continue using the camera.

Since API level 14, camera is automatically locked for applications in `start()` (/reference/android/media/MediaRecorder.html#start()). Applications can use the camera (ex: zoom) after recording starts. There is no need to call this after recording starts or stops.

If you are not recording video, you probably do not need this method.

Throws

IOException	if a connection cannot be re-established (for example, if the camera is still in use by another process).

public final void **release** ()

Added in API level 1

Disconnects and releases the Camera object resources.

You must call this as soon as you're done with the Camera object.

public void **setAutoFocusMoveCallback** (Camera.AutoFocusMoveCallback cb)

Added in API level 16

Sets camera auto-focus move callback.

Parameters

cb	the callback to run

public final void **setDisplayOrientation** (int degrees)

Added in API level 8

Set the clockwise rotation of preview display in degrees. This affects the preview frames and the picture displayed after snapshot. This method is useful for portrait mode applications. Note that preview display of front-facing cameras is flipped horizontally before the rotation, that is, the image is reflected along the central vertical axis of the camera sensor. So the users can see themselves as looking into a mirror.

This does not affect the order of byte array passed in `onPreviewFrame(byte[], Camera)` `(/reference/android/hardware/Camera.PreviewCallback.html#onPreviewFrame(byte[], android.hardware.Camera))`, JPEG pictures, or recorded videos. This method is not allowed to be called during preview.

If you want to make the camera image show in the same orientation as the display, you can use the following code.

```
public static void setCameraDisplayOrientation(Activity activity,
        int cameraId, android.hardware.Camera camera) {
    android.hardware.Camera.CameraInfo info =
            new android.hardware.Camera.CameraInfo();
    android.hardware.Camera.getCameraInfo(cameraId, info);
    int rotation = activity.getWindowManager().getDefaultDisplay()
            .getRotation();
    int degrees = 0;
    switch (rotation) {
        case Surface.ROTATION_0: degrees = 0; break;
        case Surface.ROTATION_90: degrees = 90; break;
        case Surface.ROTATION_180: degrees = 180; break;
        case Surface.ROTATION_270: degrees = 270; break;
    }

    int result;
    if (info.facing == Camera.CameraInfo.CAMERA_FACING_FRONT) {
        result = (info.orientation + degrees) % 360;
        result = (360 - result) % 360;  // compensate the mirror
    } else {  // back-facing
        result = (info.orientation - degrees + 360) % 360;
    }
    camera.setDisplayOrientation(result);
}
```

Starting from API level 14, this method can be called when preview is active.

Parameters

degrees	the angle that the picture will be rotated clockwise. Valid values are 0, 90, 180, and 270. The starting position is 0 (landscape).

See Also

`setPreviewDisplay(SurfaceHolder)`

public final void **setErrorCallback** (Camera.ErrorCallback cb)

Added in API level 1

Registers a callback to be invoked when an error occurs.

Parameters

cb	The callback to run

public final void **setFaceDetectionListener** (Camera.FaceDetectionListener listener)

Added in API level 14

Registers a listener to be notified about the faces detected in the preview frame.

Parameters

listener	the listener to notify

See Also

`startFaceDetection()`

public final void **setOneShotPreviewCallback** (Camera.PreviewCallback cb)

Added in API level 3

Installs a callback to be invoked for the next preview frame in addition to displaying it on the screen. After one invocation, the callback is cleared. This method can be called any time, even when preview is live. Any other preview callbacks are overridden.

If you are using the preview data to create video or still images, strongly consider using `MediaActionSound` `(/reference/android/media/MediaActionSound.html)` to properly indicate image capture or recording start/stop to the user.

Parameters

cb	a callback object that receives a copy of the next preview frame, or null to stop receiving callbacks.

See Also

`MediaActionSound`

public void **setParameters** (Camera.Parameters params)

Added in API level 1

Changes the settings for this Camera service.

Parameters

params	the Parameters to use for this Camera service

Throws

RuntimeException	if any parameter is invalid or not supported.

See Also

`getParameters()`

public final void **setPreviewCallback** (Camera.PreviewCallback cb)

Added in API level 1

Installs a callback to be invoked for every preview frame in addition to displaying them on the screen. The callback will be repeatedly called for as long as preview is active. This method can be called at any time, even while preview is live. Any other preview callbacks are overridden.

If you are using the preview data to create video or still images, strongly consider using `MediaActionSound` (/reference/android/media/MediaActionSound.html) to properly indicate image capture or recording start/stop to the user.

Parameters

cb	a callback object that receives a copy of each preview frame, or null to stop receiving callbacks.

See Also

`MediaActionSound`

public final void **setPreviewCallbackWithBuffer** (Camera.PreviewCallback cb)

Added in API level 8

Installs a callback to be invoked for every preview frame, using buffers supplied with `addCallbackBuffer(byte[])` (/reference/android/hardware/Camera.html#addCallbackBuffer(byte[])), in addition to displaying them on the screen. The callback will be repeatedly called for as long as preview is active and buffers are available. Any other preview callbacks are overridden.

The purpose of this method is to improve preview efficiency and frame rate by allowing preview frame memory reuse. You must call `addCallbackBuffer(byte[])` (/reference/android/hardware/Camera.html#addCallbackBuffer(byte[])) at some point -- before or after calling this method -- or no callbacks will received.

The buffer queue will be cleared if this method is called with a null callback, `setPreviewCallback(Camera.PreviewCallback)` (/reference/android/hardware/Camera.html#setPreviewCallback(android.hardware.Camera.PreviewCallback)) is called, or `setOneShotPreviewCallback(Camera.PreviewCallback)` (/reference/android/hardware/Camera.html#setOneShotPreviewCallback(android.hardware.Camera.PreviewCallback)) is called.

If you are using the preview data to create video or still images, strongly consider using `MediaActionSound` (/reference/android/media/MediaActionSound.html) to properly indicate image capture or recording start/stop to the user.

Parameters

cb	a callback object that receives a copy of the preview frame, or null to stop receiving callbacks and clear the buffer queue.

See Also

`addCallbackBuffer(byte[])`
`MediaActionSound`

public final void **setPreviewDisplay** (SurfaceHolder holder)

Added in API level 1

Sets the `Surface` (/reference/android/view/Surface.html) to be used for live preview. Either a surface or surface texture is necessary for preview, and preview is necessary to take pictures. The same surface can be re-set without harm. Setting a preview surface will un-set any preview surface texture that was set via `setPreviewTexture(SurfaceTexture)` (/reference/android/hardware/Camera.html#setPreviewTexture(android.graphics.SurfaceTexture)).

The `SurfaceHolder` (/reference/android/view/SurfaceHolder.html) must already contain a surface when this method is called. If you are using `SurfaceView` (/reference/android/view/SurfaceView.html), you will need to register a `SurfaceHolder.Callback` (/reference/android/view/SurfaceHolder.Callback.html) with `addCallback(SurfaceHolder.Callback)` (/reference/android/view/SurfaceHolder.html#addCallback(android.view.SurfaceHolder.Callback)) and wait for `surfaceCreated(SurfaceHolder)` (/reference/android/view/SurfaceHolder.Callback.html#surfaceCreated(android.view.SurfaceHolder)) before calling setPreviewDisplay() or starting preview.

This method must be called before `startPreview()` (/reference/android/hardware/Camera.html#startPreview()). The one exception is that if the preview surface is not set (or set to null) before startPreview() is called, then this method may be called once with a non-null parameter to set the preview surface. (This allows camera setup and surface creation to happen in parallel, saving time.) The preview surface may not otherwise change while preview is running.

Parameters

holder	containing the Surface on which to place the preview, or null to remove the preview surface

Throws

IOException	if the method fails (for example, if the surface is unavailable or unsuitable).

public final void **setPreviewTexture** (SurfaceTexture surfaceTexture)

Added in API level 11

Sets the `SurfaceTexture` (/reference/android/graphics/SurfaceTexture.html) to be used for live preview. Either a surface or surface texture is necessary for preview, and preview is necessary to take pictures. The same surface texture can be re-set without harm. Setting a preview surface texture will un-set any preview surface that was set via `setPreviewDisplay(SurfaceHolder)` (/reference/android/hardware/Camera.html#setPreviewDisplay(android.view.SurfaceHolder)).

This method must be called before `startPreview()` (/reference/android/hardware/Camera.html#startPreview()). The one exception is that if the preview surface texture is not set (or set to null) before startPreview() is called, then this method may be called once with a non-null parameter to set the preview surface. (This allows camera setup and surface creation to happen in parallel, saving time.) The preview surface texture may not otherwise change while preview is running.

The timestamps provided by `getTimestamp()` (/reference/android/graphics/SurfaceTexture.html#getTimestamp()) for a SurfaceTexture set as the preview texture have an unspecified zero point, and cannot be directly compared between different cameras or different instances of the same camera, or across multiple runs of the same program.

If you are using the preview data to create video or still images, strongly consider using `MediaActionSound` (/reference/android/media/MediaActionSound.html) to properly indicate image capture or recording start/stop to the user.

Parameters

surfaceTexture	the `SurfaceTexture` to which the preview images are to be sent or null to remove the current preview surface texture

Throws

IOException	if the method fails (for example, if the surface texture is unavailable or unsuitable).

See Also

`MediaActionSound`
`SurfaceTexture`
`TextureView`

public final void **setZoomChangeListener** (Camera.OnZoomChangeListener listener)

Added in API level 8

Registers a listener to be notified when the zoom value is updated by the camera driver during smooth zoom.

Parameters

listener	the listener to notify

See Also

`startSmoothZoom(int)`

public final void **startFaceDetection** ()

Added in API level 14

Starts the face detection. This should be called after preview is started. The camera will notify `Camera.FaceDetectionListener` (/reference/android/hardware/Camera.FaceDetectionListener.html) of the detected faces in the preview frame. The detected faces may be the same as the previous ones. Applications should call `stopFaceDetection()` (/reference/android/hardware/Camera.html#stopFaceDetection()) to stop the face detection. This method is supported if `getMaxNumDetectedFaces()` (/reference/android/hardware/Camera.Parameters.html#getMaxNumDetectedFaces()) returns a number larger than 0. If the face detection has started, apps should not call this again.

When the face detection 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>)), and `setMeteringAreas(List)` (/reference/android/hardware/Camera.Parameters.html#setMeteringAreas(java.util.List<android.hardware.Camera.Area>)) have no effect. The camera uses the detected faces to do auto-white balance, auto exposure, and autofocus.

If the apps call `autoFocus(AutoFocusCallback)` (/reference/android/hardware/Camera.html#autoFocus(android.hardware.Camera.AutoFocusCallback)), the camera will stop sending face callbacks. The last face callback indicates the areas used to do autofocus. After focus completes, face detection will resume sending face callbacks. If the apps call `cancelAutoFocus()` (/reference/android/hardware/Camera.html#cancelAutoFocus()), the face callbacks will also resume.

After calling `takePicture(Camera.ShutterCallback,` Camera.PictureCallback, Camera.PictureCallback) (/reference/android/hardware/Camera.html#takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) or stopPreview() (/reference/android/hardware/Camera.html#stopPreview()), and then resuming preview with startPreview() (/reference/android/hardware/Camera.html#startPreview()), the apps should call this method again to resume face detection.

Throws

IllegalArgumentException	if the face detection is unsupported.
RuntimeException	if the method fails or the face detection is already running.

See Also

Camera.FaceDetectionListener
stopFaceDetection()
getMaxNumDetectedFaces()

public final void **startPreview** ()

Added in API level 1

Starts capturing and drawing preview frames to the screen. Preview will not actually start until a surface is supplied with setPreviewDisplay(SurfaceHolder) (/reference/android/hardware/Camera.html#setPreviewDisplay(android.view.SurfaceHolder)) or setPreviewTexture(SurfaceTexture) (/reference/android/hardware/Camera.html#setPreviewTexture(android.graphics.SurfaceTexture)).

If setPreviewCallback(Camera.PreviewCallback) (/reference/android/hardware/Camera.html#setPreviewCallback(android.hardware.Camera.PreviewCallback)), setOneShotPreviewCallback(Camera.PreviewCallback) (/reference/android/hardware/Camera.html#setOneShotPreviewCallback(android.hardware.Camera.PreviewCallback)), or setPreviewCallbackWithBuffer(Camera.PreviewCallback) (/reference/android/hardware/Camera.html#setPreviewCallbackWithBuffer(android.hardware.Camera.PreviewCallback)) were called, onPreviewFrame(byte[], Camera) (/reference/android/hardware/Camera.PreviewCallback.html#onPreviewFrame(byte[], android.hardware.Camera)) will be called when preview data becomes available.

public final void **startSmoothZoom** (int value)

Added in API level 8

Zooms to the requested value smoothly. The driver will notify Camera.OnZoomChangeListener (/reference/android/hardware/Camera.OnZoomChangeListener.html) of the zoom value and whether zoom is stopped at the time. For example, suppose the current zoom is 0 and startSmoothZoom is called with value 3. The onZoomChange(int, boolean, Camera) (/reference/android/hardware/Camera.OnZoomChangeListener.html#onZoomChange(int, boolean, android.hardware.Camera)) method will be called three times with zoom values 1, 2, and 3. Applications can call stopSmoothZoom() (/reference/android/hardware/Camera.html#stopSmoothZoom()) to stop the zoom earlier. Applications should not call startSmoothZoom again or change the zoom value before zoom stops. If the supplied zoom value equals to the current zoom value, no zoom callback will be generated. This method is supported if isSmoothZoomSupported() (/reference/android/hardware/Camera.Parameters.html#isSmoothZoomSupported()) returns true.

Parameters

value	zoom value. The valid range is 0 to `getMaxZoom()`.

Throws

IllegalArgumentException	if the zoom value is invalid.
RuntimeException	if the method fails.

See Also

`setZoomChangeListener(OnZoomChangeListener)`

public final void **stopFaceDetection** ()

Added in API level 14

Stops the face detection.

See Also

`startFaceDetection()`

public final void **stopPreview** ()

Added in API level 1

Stops capturing and drawing preview frames to the surface, and resets the camera for a future call to `startPreview()` `(/reference/android/hardware/Camera.html#startPreview())`.

public final void **stopSmoothZoom** ()

Added in API level 8

Stops the smooth zoom. Applications should wait for the `Camera.OnZoomChangeListener` `(/reference/android/hardware/Camera.OnZoomChangeListener.html)` to know when the zoom is actually stopped. This method is supported if `isSmoothZoomSupported()` `(/reference/android/hardware/Camera.Parameters.html#isSmoothZoomSupported())` is true.

Throws

RuntimeException	if the method fails.

public final void **takePicture** (Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)

Added in API level 1

Equivalent to takePicture(shutter, raw, null, jpeg).

See Also

`takePicture(ShutterCallback, PictureCallback, PictureCallback, PictureCallback)`

public final void **takePicture** (Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback postview, Camera.PictureCallback jpeg)

Added in API level 5

Triggers an asynchronous image capture. The camera service will initiate a series of callbacks to the application as the image capture progresses. The shutter callback occurs after the image is captured. This can be used to trigger a sound to let the user know that image has been captured. The raw callback occurs when the raw image data is available (NOTE: the data will be null if there is no raw image callback buffer available or the raw image callback buffer is not large enough to hold the raw image). The postview callback occurs when a scaled, fully processed postview image is available (NOTE: not all hardware supports this). The jpeg callback occurs when the compressed image is available. If the application does not need a particular callback, a null can be passed instead of a callback method.

This method is only valid when preview is active (after `startPreview()` (/reference/android/hardware/Camera.html#startPreview())). Preview will be stopped after the image is taken; callers must call `startPreview()` (/reference/android/hardware/Camera.html#startPreview()) again if they want to re-start preview or take more pictures. This should not be called between `start()` (/reference/android/media/MediaRecorder.html#start()) and `stop()` (/reference/android/media/MediaRecorder.html#stop()).

After calling this method, you must not call `startPreview()` (/reference/android/hardware/Camera.html#startPreview()) or take another picture until the JPEG callback has returned.

Parameters

shutter	the callback for image capture moment, or null
raw	the callback for raw (uncompressed) image data, or null
postview	callback with postview image data, may be null
jpeg	the callback for JPEG image data, or null

public final void **unlock** ()

Added in API level 5

Unlocks the camera to allow another process to access it. Normally, the camera is locked to the process with an active Camera object until `release()` (/reference/android/hardware/Camera.html#release()) is called. To allow rapid handoff between processes, you can call this method to release the camera temporarily for another process to use; once the other process is done you can call `reconnect()` (/reference/android/hardware/Camera.html#reconnect()) to reclaim the camera.

This must be done before calling `setCamera(Camera)` (/reference/android/media/MediaRecorder.html#setCamera(android.hardware.Camera)). This cannot be called after recording starts.

If you are not recording video, you probably do not need this method.

Throws

RuntimeException	if the camera cannot be unlocked.

Protected Methods

protected void **finalize** ()

Added in API level 1

Invoked when the garbage collector has detected that this instance is no longer reachable. The default implementation does nothing, but this method can be overridden to free resources.

Note that objects that override `finalize` are significantly more expensive than objects that don't. Finalizers may be run a long time after the object is no longer reachable, depending on memory pressure, so it's a bad idea to rely on them for cleanup. Note also that finalizers are run on a single VM-wide finalizer thread, so doing blocking work in a finalizer is a bad idea. A finalizer is usually only necessary for a class that has a native peer and needs to call a native method to destroy that peer. Even then, it's better to provide an explicit `close` method (and implement `Closeable` (/reference/java/io/Closeable.html)), and insist that callers manually dispose of instances. This works well for something like files, but less well for something like a `BigInteger` where typical calling code would have to deal with lots of temporaries. Unfortunately, code that creates lots of temporaries is the worst kind of code from the point of view of the single finalizer thread.

If you *must* use finalizers, consider at least providing your own `ReferenceQueue` (/reference/java/lang/ref/ReferenceQueue.html) and having your own thread process that queue.

Unlike constructors, finalizers are not automatically chained. You are responsible for calling `super.finalize()` yourself.

Uncaught exceptions thrown by finalizers are ignored and do not terminate the finalizer thread. See *Effective Java* Item 7, "Avoid finalizers" for more.