Utilizing the Camera with Android

As with most other functionality on Android devices, there are multiple ways to utilize the camera.

Capturing a picture with an Intent

The easiest way but perhaps least flexible is to create an Intent to launch the built-in camera Activity and return the resulting picture to your Activity. Unfortunately, this isn't without it's problems. (For a variety of reasons, Android doesn't return a full size image using this method.)

The version below may only work correctly on 2.0 devices and higher. For earlier devices, Android may not be able to write the image to a file based on an Intent.

```
/*
 * Make sure your emulator has an "SD Card" size specified or just test on a device
 */

package com.mobvcasting.pictureintent;

import java.io.File;
import java.io.FileDescriptor;
import java.io.FileNotFoundException;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;

public class PictureIntent extends Activity {

     Button aButton;
     String imageFilePath;

     ImageView ourImageView;

     public final int CAMERA_RESULT = 0;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        imageFilePath = Environment.getExternalStorageDirectory().getAbsolutePath() + "/tmp_image.jpg";
        Log.v("IMAGE FILE",imageFilePath);

        aButton = (Button) this.findViewById(R.id.Button01);
        aButton.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {

                     File imageFile = new File(imageFilePath);
                     Uri imageFileUri = Uri.fromFile(imageFile);


                     Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
                     i.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);

                     startActivityForResult(i, CAMERA_RESULT);
                }});

        ourImageView = (ImageView) this.findViewById(R.id.ImageView01);
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent intent)
    {
          super.onActivityResult(requestCode, resultCode, intent);

          //Bundle extras = intent.getExtras();

          switch(requestCode)
          {
               case CAMERA_RESULT:
```

```
                    if (resultCode == RESULT_OK)
                    {
                         Log.v("RESULTS","HERE");

                         /* Tiny Image Returned
                         Bitmap bmp = (Bitmap) extras.get("data");
                         ourImageView.setImageBitmap(bmp);
                         */

                         Bitmap bmp = BitmapFactory.decodeFile(imageFilePath);
                         ourImageView.setImageBitmap(bmp);

                         Log.v("RESULTS","Image Width: " + bmp.getWidth());
                         Log.v("RESULTS","Image Height: " + bmp.getHeight());
                    }
                    break;
          }
    }
}
```

This requires that we request some permissions in our AndroidManifest.xml file:

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

Capturing a picture

Another way to capture an image with Android is to create our own camera application. This is more difficult but should enable the grabbing of a full size image. It also allows for complex interactions with the camera preview image.

The first issue we run into is that we need to create a SurfaceView for the camera to draw the preview image onto. This preview image will act as our view finder so that the user can see what they are about to take a picture of.

Here is a barebones SurfaceView that we can get started with:

```
package com.mobvcasting.snapshot;

import android.content.Context;
import android.util.AttributeSet;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class CameraView extends SurfaceView implements SurfaceHolder.Callback
{
    SurfaceHolder mHolder;

    public CameraView(Context context, AttributeSet attrs)
    {
        super(context,attrs);
        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public CameraView(Context context)
    {
        super(context);
        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder)
    {
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
    }

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

SurfaceView's can be used in within Layout XML files. To put this bare bones SurfaceView within an Activity that will display it, we would use the following XML:

```
<com.mobvcasting.snapshot.CameraView android:id="@+id/CameraView01" android:layout_width="wrap_content"
android:layout_height="wrap_content"></com.mobvcasting.snapshot.CameraView>
```

This SurfaceView implements something called a SurfaceHolder.Callback. This allows our SurfaceView to be notified by calls when it is created, destroyed or changed. To ask for this notification we have to get a reference to our "Holder" and add ourself as the callback listener:

```
        mHolder = getHolder();
        mHolder.addCallback(this);
```

This is done in our constructor.

Now our methods for surfaceCreated, surfaceDestroyed and surfaceChanged should be called at the appropriate times.

Within those methods is where we want to actually deal with initializing and displaying the image from the Camera.

```
    public void surfaceCreated(SurfaceHolder holder)
    {
        // The Surface has been created, acquire the camera and tell it where to draw.
        mCamera = Camera.open();

        try
        {
           mCamera.setPreviewDisplay(holder);
        }
        catch (IOException exception)
        {
            mCamera.release();
            mCamera = null;
        }
    }
```

In surfaceCreated, as shown above we want to open the camera and set it's preview to be our holder.

In surfaceDestroyed we want to release the camera:

```
    public void surfaceDestroyed(SurfaceHolder holder) {
        // Surface will be destroyed when we return, so stop the preview.
        // Because the CameraDevice object is not a shared resource, it's very
        // important to release it when the activity is paused.
        mCamera.stopPreview();
        mCamera.release();
        mCamera = null;
    }
```

In surfaceChanged we want to start the preview:

```
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
    {
        mCamera.startPreview();
    }
```

We also need a method to take a picture with the camera. Here is one that simply wraps the method provided by the Camera object:

```
    public void takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)
    {
     mCamera.takePicture(shutter, raw, jpeg);
    }
```

The Camera class is Android's main class for dealing with the still camera hardware of any device. In order to use the camera we have to ask for permission in our AndroidManifest.xml file:

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

Here is the full code to our CameraView SurfaceView class:

```
package com.mobvcasting.snapshot;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

import android.content.Context;
import android.content.res.Configuration;
import android.hardware.Camera;
import android.hardware.Camera.Parameters;
import android.util.AttributeSet;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class CameraView extends SurfaceView implements SurfaceHolder.Callback
{
    SurfaceHolder mHolder;

    int width;
    int height;

    Camera mCamera;

    public CameraView(Context context, AttributeSet attrs)
    {
     super(context,attrs);
        holderCreation();
    }

    public CameraView(Context context)
    {
        super(context);
        holderCreation();
    }

    public void takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)
    {
     mCamera.takePicture(shutter, raw, jpeg);
    }

    public void holderCreation()
    {
        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder)
    {
        // The Surface has been created, acquire the camera and tell it where to draw.
        mCamera = Camera.open();

        Parameters params = mCamera.getParameters();

        // If we aren't landscape (the default), tell the camera we want portrait mode
        if (this.getResources().getConfiguration().orientation !=
          Configuration.ORIENTATION_LANDSCAPE)
        {
            params.set("orientation", "portrait"); // "landscape"

            // And Rotate the final picture if possible
            // This works on 2.0 and higher only
            //params.setRotation(90);
            // Use reflection to see if it exists and to call it so you can support older versions
            try {
                Method rotateSet = Camera.Parameters.class.getMethod(
                        "setRotation", new Class[] { Integer.TYPE }  );
                Object arguments[] = new Object[] { new Integer(90) };
                rotateSet.invoke(params, arguments);
            } catch (NoSuchMethodException nsme) {
                // Older Device
                Log.v("CAMERAVIEW","No Set Rotation");
            } catch (IllegalArgumentException e) {
                Log.v("CAMERAVIEW","Exception IllegalArgument");
          } catch (IllegalAccessException e) {
                Log.v("CAMERAVIEW","Illegal Access Exception");
          } catch (InvocationTargetException e) {
                Log.v("CAMERAVIEW","Invocation Target Exception");
          }
        }
```

```
        mCamera.setParameters(params);

        try
        {
           mCamera.setPreviewDisplay(holder);
        }
        catch (IOException exception)
        {
            mCamera.release();
            mCamera = null;
        }
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        // Surface will be destroyed when we return, so stop the preview.
        // Because the CameraDevice object is not a shared resource, it's very
        // important to release it when the activity is paused.
        mCamera.stopPreview();
        mCamera.release();
        mCamera = null;
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
    {
     width = w;
     height = h;

        // Now that the size is known, set up the camera parameters and begin the preview.
        Camera.Parameters parameters = mCamera.getParameters();
        //parameters.setPreviewSize(w, h);
        mCamera.setParameters(parameters);
        mCamera.startPreview();
    }
}
```

Next we'll need an Activity that displays the CameraView and has a button or something similar to trigger the taking of the picture.

This activity will also implement Camera.PictureCallback so it gets notified when the picture taking is complete:

```
package com.mobvcasting.snapshot;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Bitmap.CompressFormat;
import android.hardware.Camera;

public class SnapShot extends Activity implements OnClickListener, Camera.PictureCallback {

     CameraView cameraView;
     ImageView imv;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        setContentView(R.layout.main);

        Button pictureButton = (Button) this.findViewById(R.id.Button01);
        imv = (ImageView) this.findViewById(R.id.ImageView01);
        cameraView = (CameraView) this.findViewById(R.id.CameraView01);

        pictureButton.setOnClickListener(this);

    }

    // From the OnClickListener
```

```
    public void onClick(View v)
    {
     cameraView.takePicture(null, null, this);
    }

    // From the Camera.PictureCallback
    public void onPictureTaken(byte[] data, Camera camera)
    {
           Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
           imv.setImageBitmap(bmp);

           String filename = "apicture.jpg";
        File pictureFile = new File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/" + filename);
        try
        {
               FileOutputStream pfos = new FileOutputStream(pictureFile);
          bmp.compress(CompressFormat.JPEG, 75, pfos);

          pfos.flush();
          pfos.close();
          }
        catch (FileNotFoundException e)
          {
               e.printStackTrace();
          }
        catch (IOException e)
        {
               e.printStackTrace();
          }

    }
}
```

In the layout XML for this activity we have the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<Button android:id="@+id/Button01" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Take
Picture"></Button>

<ImageView android:id="@+id/ImageView01" android:layout_width="wrap_content" android:layout_height="wrap_content"></ImageView>
<com.mobvcasting.snapshot.CameraView android:id="@+id/CameraView01" android:layout_width="wrap_content"
android:layout_height="wrap_content"></com.mobvcasting.snapshot.CameraView>
</LinearLayout>
```

It includes a Button for taking the picture, an ImageView for displaying the resulting image and our CameraView.

In our code we get references to these as follows in our onCreate method:

```
        Button pictureButton = (Button) this.findViewById(R.id.Button01);
        imv = (ImageView) this.findViewById(R.id.ImageView01);
        cameraView = (CameraView) this.findViewById(R.id.CameraView01);
```

Later, you see that the pictureButton's onClickListener is set to "this" which means that our "onClick" method will be called when the button is pushed.

```
 pictureButton.setOnClickListener(this);
```

That method simply calls the "takePicture" method in the cameraView object, passing in null for the first two options and "this" for the last which is the Camera.PictureCallback.

When the picture is taken, our Activity's onPictureTaken method gets called. This method decodes the data from the camera into a Bitmap object and sets the ImageView object's bitmap to be it. This all just displays the image.

```
          Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
          imv.setImageBitmap(bmp);
```

Next, a File object is created and the picture is compressed as a JPEG and written to that file.

```
String filename = "apicture.jpg";
File pictureFile = new File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/" + filename);

FileOutputStream pfos = new FileOutputStream(pictureFile);
bmp.compress(CompressFormat.JPEG, 75, pfos);

pfos.flush();
pfos.close();
```

This requires that we have permission to write to the SD card:

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

If you explore the Snapshot example, you'll see that it also implements uploading the captured image to the internet.