[**Live camera preview in the Android emulator**](http://www.inter-fuser.com/2009/09/live-camera-preview-in-android-emulator.html)

I've been looking into getting a live camera preview working in the Android emulator. Currently the Android emulator just gives a black and white chess board animation. After having a look around I found the web site of Tom Gibara who has done some great work to get a live preview working in the emulator. The link to his work can be found [here](http://www.tomgibara.com/android/camera-source). The basics are that you run the WebcamBroadcaster as a standard java app on your PC. If there are any video devices attached to you PC, it will pick them up and broadcast the frames captured over a socket connection. You then run a SocketCamera class as part of an app in the android emulator, and as long as you have the correct ip address and port it should display the captured images in the emulator. On looking into Tom's code I saw that it seemed to be written for an older version of the Android API so I thought I'd have a go at updating it. As a starting point I'm going to use the  
[CameraPreview](http://developer.android.com/guide/samples/ApiDemos/src/com/example/android/apis/graphics/CameraPreview.html) sample code available on the android developers website. My aim was to take this code and with as little changes as possible make it so it could be used to give a live camera preview in the emulator.  
  
So the first thing I did was to create a new class called SocketCamera, this is based on Tom's version of the SocketCamera, but unlike Tom's version I am trying to implement a subset of the new camera class android.hardware.Camera and not the older class android.hardware.CameraDevice. Please keep in mind that I've implemented just a subset of the Camera class API. The code was put together fairly quickly and is a bit rough round the edges. Anyhow, here's my new SocketCamera class:

001.package com.example.socketcamera;

002.

003.import java.io.IOException;

004.import java.io.InputStream;

005.import java.net.InetSocketAddress;

006.import java.net.Socket;

007.

008.import android.graphics.Bitmap;

009.import android.graphics.BitmapFactory;

010.import android.graphics.Canvas;

011.import android.graphics.Paint;

012.import android.graphics.Rect;

013.import android.hardware.Camera;

014.import android.hardware.Camera.Size;

015.import android.util.Log;

016.import android.view.SurfaceHolder;

017.

018.public class SocketCamera {

019.

020.private static final String LOG\_TAG = "SocketCamera:";

021.private static final int SOCKET\_TIMEOUT = 1000;

022.

023.static private SocketCamera socketCamera;

024.private CameraCapture capture;

025.private Camera parametersCamera;

026.private SurfaceHolder surfaceHolder;

027.

028.//Set the IP address of your pc here!!

029.private final String address = "192.168.1.12";

030.private final int port = 9889;

031.

032.private final boolean preserveAspectRatio = true;

033.private final Paint paint = new Paint();

034.

035.

036.private int width = 240;

037.private int height = 320;

038.private Rect bounds = new Rect(0, 0, width, height);

039.

040.private SocketCamera() {

041.//Just used so that we can pass Camera.Paramters in getters and setters

042.parametersCamera = Camera.open();

043.}

044.

045.static public SocketCamera open()

046.{

047.if (socketCamera == null) {

048.socketCamera = new SocketCamera();

049.}

050.

051.Log.i(LOG\_TAG, "Creating Socket Camera");

052.return socketCamera;

053.}

054.

055.public void startPreview() {

056.capture = new CameraCapture();

057.capture.setCapturing(true);

058.capture.start();

059.Log.i(LOG\_TAG, "Starting Socket Camera");

060.

061.}

062.

063.public void stopPreview(){

064.capture.setCapturing(false);

065.Log.i(LOG\_TAG, "Stopping Socket Camera");

066.}

067.

068.public void setPreviewDisplay(SurfaceHolder surfaceHolder) throws IOException {

069.this.surfaceHolder = surfaceHolder;

070.}

071.

072.public void setParameters(Camera.Parameters parameters) {

073.//Bit of a hack so the interface looks like that of

074.Log.i(LOG\_TAG, "Setting Socket Camera parameters");

075.parametersCamera.setParameters(parameters);

076.Size size = parameters.getPreviewSize();

077.bounds = new Rect(0, 0, size.width, size.height);

078.}

079.public Camera.Parameters getParameters() {

080.Log.i(LOG\_TAG, "Getting Socket Camera parameters");

081.return parametersCamera.getParameters();

082.}

083.

084.public void release() {

085.Log.i(LOG\_TAG, "Releasing Socket Camera parameters");

086.//TODO need to implement this function

087.}

088.

089.

090.private class CameraCapture extends Thread  {

091.

092.private boolean capturing = false;

093.

094.public boolean isCapturing() {

095.return capturing;

096.}

097.

098.public void setCapturing(boolean capturing) {

099.this.capturing = capturing;

100.}

101.

102.@Override

103.public void run() {

104.while (capturing) {

105.Canvas c = null;

106.try {

107.c = surfaceHolder.lockCanvas(null);

108.synchronized (surfaceHolder) {

109.Socket socket = null;

110.try {

111.socket = new Socket();

112.socket.bind(null);

113.socket.setSoTimeout(SOCKET\_TIMEOUT);

114.socket.connect(new InetSocketAddress(address, port), SOCKET\_TIMEOUT);

115.

116.//obtain the bitmap

117.InputStream in = socket.getInputStream();

118.Bitmap bitmap = BitmapFactory.decodeStream(in);

119.

120.//render it to canvas, scaling if necessary

121.if (

122.bounds.right == bitmap.getWidth() &&

123.bounds.bottom == bitmap.getHeight()) {

124.c.drawBitmap(bitmap, 0, 0, null);

125.} else {

126.Rect dest;

127.if (preserveAspectRatio) {

128.dest = new Rect(bounds);

129.dest.bottom = bitmap.getHeight() \* bounds.right / bitmap.getWidth();

130.dest.offset(0, (bounds.bottom - dest.bottom)/2);

131.} else {

132.dest = bounds;

133.}

134.if (c != null)

135.{

136.c.drawBitmap(bitmap, null, dest, paint);

137.}

138.}

139.

140.} catch (RuntimeException e) {

141.e.printStackTrace();

142.

143.} catch (IOException e) {

144.e.printStackTrace();

145.} finally {

146.try {

147.socket.close();

148.} catch (IOException e) {

149./\* ignore \*/

150.}

151.}

152.}

153.} catch (Exception e) {

154.e.printStackTrace();

155.} finally {

156.

157.// do this in a finally so that if an exception is thrown

158.// during the above, we don't leave the Surface in an

159.// inconsistent state

160.if (c != null) {

161.surfaceHolder.unlockCanvasAndPost(c);

162.}

163.}

164.}

165.Log.i(LOG\_TAG, "Socket Camera capture stopped");

166.}

167.}

168.

169.}

Make sure that you change the ip address to that of your PC.  
  
Now we just need to make a few small modifications to the original CameraPreview. In this class look for the Preview class that extends the SurfaceView. Now we just need to comments out three lines and replace them with our own:

01.class Preview extends SurfaceView implements SurfaceHolder.Callback {

02.SurfaceHolder mHolder;

03.//Camera mCamera;

04.SocketCamera mCamera;

05.Preview(Context context) {

06.super(context);

07.

08.// Install a SurfaceHolder.Callback so we get notified when the

09.// underlying surface is created and destroyed.

10.mHolder = getHolder();

11.mHolder.addCallback(this);

12.//mHolder.setType(SurfaceHolder.SURFACE\_TYPE\_PUSH\_BUFFERS);

13.mHolder.setType(SurfaceHolder.SURFACE\_TYPE\_NORMAL);

14.}

15.public void surfaceCreated(SurfaceHolder holder) {

16.// The Surface has been created, acquire the camera and tell it where

17.// to draw.

18.//mCamera = Camera.open();

19.mCamera = SocketCamera.open();

20.try {

21.mCamera.setPreviewDisplay(holder);

22.} catch (IOException exception) {

23.mCamera.release();

24.mCamera = null;

25.// TODO: add more exception handling logic here

26.}

27.}

Here i've change three lines:  
  
1. Camera mCamera is replaced with SocketCamera mCamera  
2. mHolder.setType(SurfaceHolder.SURFACE\_TYPE\_PUSH\_BUFFERS); is replaced with mHolder.setType(SurfaceHolder.SURFACE\_TYPE\_NORMAL);  
3. mCamera = Camera.open(); is replaced with mCamera = SocketCamera.open();.  
  
So that's it.Now just make sure WebcamBroadcaster is running and start up the CameraPreview app in the Android emulator, you should now be seeing live previews in the emulator. Here's a short video of my emulator with the live preview: (Yes i know, it's me waving a book around)  
  
  
  
Note: if the WebcamBroadcaster is not picking up your devices you most probably have a classpath issue. Make sure that you classpath points to the jmf.jar that is in the same folder as the jmf.properties file. If JMstudio works ok, its very likely that you have a classpath issue.  
  
Oh, one last thing. I also updated the WebCamBroadcaster so that it can be used with YUV format cameras, so here's the code for that as well:

001.package com.webcambroadcaster;

002.import java.awt.Dimension;

003.import java.awt.image.BufferedImage;

004.import java.io.BufferedOutputStream;

005.import java.io.DataOutputStream;

006.import java.io.IOException;

007.import java.io.OutputStream;

008.import java.net.ServerSocket;

009.import java.net.Socket;

010.import java.util.Vector;

011.

012.import javax.imageio.ImageIO;

013.import javax.media.Buffer;

014.import javax.media.CannotRealizeException;

015.import javax.media.CaptureDeviceInfo;

016.import javax.media.CaptureDeviceManager;

017.import javax.media.Format;

018.import javax.media.Manager;

019.import javax.media.MediaLocator;

020.import javax.media.NoDataSourceException;

021.import javax.media.NoPlayerException;

022.import javax.media.Player;

023.import javax.media.control.FrameGrabbingControl;

024.import javax.media.format.RGBFormat;

025.import javax.media.format.VideoFormat;

026.import javax.media.format.YUVFormat;

027.import javax.media.protocol.CaptureDevice;

028.import javax.media.protocol.DataSource;

029.import javax.media.util.BufferToImage;

030.

031./\*\*

032.\* A disposable class that uses JMF to serve a still sequence captured from a

033.\* webcam over a socket connection. It doesn't use TCP, it just blindly

034.\* captures a still, JPEG compresses it, and pumps it out over any incoming

035.\* socket connection.

036.\*

037.\* @author Tom Gibara

038.\*

039.\*/

040.

041.public class WebcamBroadcaster {

042.

043.public static boolean RAW = false;

044.

045.

046.private static Player createPlayer(int width, int height) {

047.try {

048.Vector<CaptureDeviceInfo> devices = CaptureDeviceManager.getDeviceList(null);

049.for (CaptureDeviceInfo info : devices) {

050.DataSource source;

051.Format[] formats = info.getFormats();

052.for (Format format : formats) {

053.if ((format instanceof RGBFormat)) {

054.RGBFormat rgb = (RGBFormat) format;

055.Dimension size = rgb.getSize();

056.if (size.width != width || size.height != height) continue;

057.if (rgb.getPixelStride() != 3) continue;

058.if (rgb.getBitsPerPixel() != 24) continue;

059.if ( rgb.getLineStride() != width\*3 ) continue;

060.MediaLocator locator = info.getLocator();

061.source = Manager.createDataSource(locator);

062.source.connect();

063.System.out.println("RGB Format Found");

064.((CaptureDevice)source).getFormatControls()[0].setFormat(rgb);

065.} else if ((format instanceof YUVFormat)) {

066.YUVFormat yuv = (YUVFormat) format;

067.Dimension size = yuv.getSize();

068.if (size.width != width || size.height != height) continue;

069.MediaLocator locator = info.getLocator();

070.source = Manager.createDataSource(locator);

071.source.connect();

072.System.out.println("YUV Format Found");

073.((CaptureDevice)source).getFormatControls()[0].setFormat(yuv);

074.} else {

075.continue;

076.}

077.

078.return Manager.createRealizedPlayer(source);

079.}

080.}

081.} catch (IOException e) {

082.System.out.println(e.toString());

083.e.printStackTrace();

084.} catch (NoPlayerException e) {

085.System.out.println(e.toString());

086.e.printStackTrace();

087.} catch (CannotRealizeException e) {

088.System.out.println(e.toString());

089.e.printStackTrace();

090.} catch (NoDataSourceException e) {

091.System.out.println(e.toString());

092.e.printStackTrace();

093.}

094.return null;

095.}

096.

097.public static void main(String[] args) {

098.int[] values = new int[args.length];

099.for (int i = 0; i < values.length; i++) {

100.values[i] = Integer.parseInt(args[i]);

101.}

102.

103.WebcamBroadcaster wb;

104.if (values.length == 0) {

105.wb = new WebcamBroadcaster();

106.} else if (values.length == 1) {

107.wb = new WebcamBroadcaster(values[0]);

108.} else if (values.length == 2) {

109.wb = new WebcamBroadcaster(values[0], values[1]);

110.} else {

111.wb = new WebcamBroadcaster(values[0], values[1], values[2]);

112.}

113.

114.wb.start();

115.}

116.

117.public static final int DEFAULT\_PORT = 9889;

118.public static final int DEFAULT\_WIDTH = 320;

119.public static final int DEFAULT\_HEIGHT = 240;

120.

121.private final Object lock = new Object();

122.

123.private final int width;

124.private final int height;

125.private final int port;

126.

127.private boolean running;

128.

129.private Player player;

130.private FrameGrabbingControl control;

131.private boolean stopping;

132.private Worker worker;

133.

134.public WebcamBroadcaster(int width, int height, int port) {

135.this.width = width;

136.this.height = height;

137.this.port = port;

138.}

139.

140.public WebcamBroadcaster(int width, int height) {

141.this(width, height, DEFAULT\_PORT);

142.}

143.

144.public WebcamBroadcaster(int port) {

145.this(DEFAULT\_WIDTH, DEFAULT\_HEIGHT, port);

146.}

147.

148.public WebcamBroadcaster() {

149.this(DEFAULT\_WIDTH, DEFAULT\_HEIGHT, DEFAULT\_PORT);

150.}

151.

152.public void start() {

153.synchronized (lock) {

154.if (running) return;

155.player = createPlayer(width, height);

156.if (player == null) {

157.System.err.println("Unable to find a suitable player");

158.return;

159.}

160.System.out.println("Starting the player");

161.player.start();

162.control = (FrameGrabbingControl) player.getControl("javax.media.control.FrameGrabbingControl");

163.worker = new Worker();

164.worker.start();

165.System.out.println("Grabbing frames");

166.running = true;

167.}

168.}

169.

170.public void stop() throws InterruptedException {

171.synchronized (lock) {

172.if (!running) return;

173.if (player != null) {

174.control = null;

175.player.stop();

176.player = null;

177.}

178.stopping = true;

179.running = false;

180.worker = null;

181.}

182.try {

183.worker.join();

184.} finally {

185.stopping = false;

186.}

187.}

188.

189.private class Worker extends Thread {

190.

191.private final int[] data = new int[width\*height];

192.

193.@Override

194.public void run() {

195.ServerSocket ss;

196.try {

197.ss = new ServerSocket(port);

198.

199.} catch (IOException e) {

200.e.printStackTrace();

201.return;

202.}

203.

204.while(true) {

205.FrameGrabbingControl c;

206.synchronized (lock) {

207.if (stopping) break;

208.c = control;

209.}

210.Socket socket = null;

211.try {

212.socket = ss.accept();

213.

214.Buffer buffer = c.grabFrame();

215.BufferToImage btoi = new BufferToImage((VideoFormat)buffer.getFormat());

216.BufferedImage image = (BufferedImage) btoi.createImage(buffer);

217.

218.if (image != null) {

219.OutputStream out = socket.getOutputStream();

220.if (RAW) {

221.image.getWritableTile(0, 0).getDataElements(0, 0, width, height, data);

222.image.releaseWritableTile(0, 0);

223.DataOutputStream dout = new DataOutputStream(new BufferedOutputStream(out));

224.for (int i = 0; i < data.length; i++) {

225.dout.writeInt(data[i]);

226.}

227.dout.close();

228.} else {

229.ImageIO.write(image, "JPEG", out);

230.}

231.}

232.

233.socket.close();

234.socket = null;

235.} catch (IOException e) {

236.e.printStackTrace();

237.} finally {

238.if (socket != null)

239.try {

240.socket.close();

241.} catch (IOException e) {

242./\* ignore \*/

243.}

244.}

245.

246.}

247.

248.try {

249.ss.close();

250.} catch (IOException e) {

251./\* ignore \*/

252.}

253.}

254.

255.}

256.

257.}