CHAPTER 9
Multimedia

Android is a rich multimedia environment. The standard Android load includes music and video players, and most commercial devices ship with these or fancier versions as well as YouTube players and more. The recipes in this chapter show you how to control some aspects of the multimedia world that Android provides.

9.1 Playing a YouTube Video

Marco Dinacci

Problem

You want to play a video from YouTube on your device.

Solution

Given a URI to play the video, create an `ACTION_VIEW` Intent with it and start a new Activity.

Discussion

Example 9-1 shows the code required to start a YouTube video with an Intent.

> For this recipe to work, the standard YouTube application (or one compatible with it) must be installed on the device.

Example 9-1. Starting a YouTube video with an Intent

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

    String video_path = "http://www.youtube.com/watch?v=opZ69P-0Jbc";
    Uri uri = Uri.parse(video_path);

    // With this line the YouTube application, if installed, will launch immediately.
    // Without it you will be prompted with a list of applications to choose from.
    uri = Uri.parse("vnd.youtube:"  + uri.getQueryParameter("v"));

    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    startActivity(intent);
}
```

The example uses a standard YouTube.com URL. The `uri.getQueryParameter("v")` is used to extract the video ID from the URI itself; in our example, the ID is `opZ69P-0Jbc`.

9.2 Capturing Video Using MediaRecorder

Marco Dinacci

Problem

You want to capture video using the built-in device camera and save it to disk.

Solution

Capture a video and record it on the phone by using the `c` class provided by the Android framework.

Discussion

The `MediaRecorder` is normally used to perform audio and/or video recording. The class has a straightforward API, but because it is based on a simple state machine, the methods must be called in the proper order to avoid `IllegalStateExceptions` from popping up.

Create a new Activity and override the `onCreate()` method with the code shown in Example 9-2.

Example 9-2. The onCreate() method of the main Activity

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.media_recorder_recipe);
// We shall take the video in landscape orientation
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

mSurfaceView = (SurfaceView) findViewById(R.id.surfaceView);
    mHolder = mSurfaceView.getHolder();
    mHolder.addCallback(this);
    mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    mToggleButton = (ToggleButton) findViewById(R.id.toggleRecordingButton);
    mToggleButton.setOnClickListener(new OnClickListener() {
            @Override
            // Toggle video recording
            public void onClick(View v) {
                    if (((ToggleButton)v).isChecked())
                            mMediaRecorder.start();
                    else {
                            mMediaRecorder.stop();
                            mMediaRecorder.reset();
                            try {
                                    initRecorder(mHolder.getSurface());
                            } catch (IOException e) {
                                    e.printStackTrace();
                            }
                    }
            }
    });
}
```

The preview frames from the camera will be displayed on a `SurfaceView`. Recording is controlled by a toggle button. After the recording is over, we stop the `MediaRecorder`. Since the `stop()` method resets all the state machine variables in order to be able to grab another video, we reset the state machine and call our `initRecorder()` method once more. `initRecorder()` is where we configure the `MediaRecorder` and the camera, as shown in Example 9-3.

Example 9-3. Setting up the MediaRecorder

```
/* Init the MediaRecorder. The order the methods are called in is vital to
 * its correct functioning.
 */
private void initRecorder(Surface surface) throws IOException {
    // It is very important to unlock the camera before calling setCamera(),
    // or it will result in a black preview
    if(mCamera == null) {
        mCamera = Camera.open();
        mCamera.unlock();
    }

    if(mMediaRecorder == null)
        mMediaRecorder = new MediaRecorder();

    mMediaRecorder.setPreviewDisplay(surface);
    mMediaRecorder.setCamera(mCamera);

    mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
    mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
    File file = createFile();

    mMediaRecorder.setOutputFile(file.getAbsolutePath());

    // No limit. Don't forget to check the space on disk.
    mMediaRecorder.setMaxDuration(-1);
    mMediaRecorder.setVideoFrameRate(15);

    mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);

    try {
        mMediaRecorder.prepare();
    } catch (IllegalStateException e) {
        // This is thrown if the previous calls are not made in the
        // proper order
        e.printStackTrace();
    }

    mInitSuccesful = true;
}
```

It is important to create and unlock a `Camera` object before the creation of a `MediaRecorder`. `setPreviewDisplay`, and `setCamera()` must be called immediately after the creation of the `MediaRecorder`. The choice of the format and output file is obligatory. Other options, if present, must be called in the order outlined in Example 9-3.

The `MediaRecorder` is best initialized when the surface has been created. We register our Activity as a `SurfaceHolder.Callback` listener in order to be notified of this and override the `surfaceCreated()` method to call our initialization code:

```
@Override
public void surfaceCreated(SurfaceHolder holder) {
    try {
        if(!mInitSuccessful)
            initRecorder(mHolder.getSurface());
    } catch (IOException e) {
        e.printStackTrace();    // Better error handling?
    }
}
```

When you're done with the surface, don't forget to release the resources, as the Camera is a shared object and may be used by other applications:

```
private void shutdown() {
    // Release MediaRecorder and especially the Camera as it's a shared
    // object that can be used by other applications
    mMediaRecorder.reset();
    mMediaRecorder.release();
    mCamera.release();
```

```
        // Once the objects have been released they can't be reused
        mMediaRecorder = null;
        mCamera = null;
    }
```

Override the `surfaceDestroyed()` method so that the preceding code can be called automatically when the user is done with the Activity:

```
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        shutdown();
    }
```

Source Download URL

The source code for this project is in the Android Cookbook repository, in the subdirectory *MediaRecorderDemo* (see "Getting and Using the Code Examples" on page 18).

9.3 Using Android's Face Detection Capability

Wagied Davids

Problem

You want to find out whether a given image contains any human faces and, if so, where they're located.

Solution

Use Android's built-in face detection capability.

Discussion

This recipe illustrates how to implement face detection in images. Face detection is a cool and fun hidden API feature of Android. In essence, face detection is the act of recognizing the parts of an image that appear to be human faces. It is part of a machine learning technique of recognizing objects using a set of features.

Note that this is not face *recognition*; it detects the parts of the image that are faces, but does not tell you who the faces belong to. Android 4.0 and later feature face recognition for unlocking the phone.

The main Activity (see Example 9-4) creates an instance of our `FaceDetectionView`. In this example, we hardcode the file to be scanned, but in real life you would probably want to capture the image using the camera, or choose the image from a gallery.