65. Video Recording and Image Capture on Android using Camera Intents

Many Android devices are equipped with at least one camera. There are a number of ways to allow the user to record video from within an Android application via these built-in cameras, but by far the easiest approach is to make use of a camera intent included with the Android operating system. This allows an application to invoke the standard Android video recording interface. When the user has finished recording, the intent will return to the application, passing through a reference to the media file containing the recorded video.

As will be demonstrated in this chapter, this approach allows video recording capabilities to be added to applications with just a few lines of code.

65.1 Checking for Camera Support

Before attempting to access the camera on an Android device, it is essential that defensive code be implemented to verify the presence of camera hardware. This is of particular importance since not all Android devices include a camera.

The presence or otherwise of a camera can be identified via a call to the *PackageManager.hasSystemFeature()* method. In order to check for the presence of a front-facing camera, the code needs to check for the presence of the *PackageManager.FEATURE_CAMERA_FRONT* feature. This can be encapsulated into the following convenience method:

```
private boolean hasCamera() {
       return (getPackageManager().hasSystemFeature(
           PackageManager.FEATURE_CAMERA_FRONT));
}
```

The presence of a camera facing away from the device screen can be similarly verified using the *PackageManager.FEATURE_CAMERA* constant. A test for whether a device has any camera can be performed by referencing *PackageManager.FEATURE_CAMERA_ANY*.

65.2 Calling the Video Capture Intent

Use of the video capture intent involves, at a minimum, the implementation of code to call the intent activity and a method to handle the return from the activity. The Android built-in video recording intent is represented by *MediaStore.ACTION_VIDEO_CAPTURE* and may be launched as follows:

```
private static final int VIDEO_CAPTURE = 101;

Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
startActivityForResult(intent, VIDEO_CAPTURE);
```

When invoked in this way, the intent will place the recorded video into a file using a default location and file name.

When the user either completes or cancels the video recording session, the *onActivityResult()* method of the calling activity will be called. This method needs to check that the request code passed through as an argument matches that specified when the intent was launched, verify that the recording session was successful and extract the path of the video media file. The corresponding *onActivityResult()* method for the above intent launch code might, therefore, be implemented as follows:

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    Uri videoUri = data.getData();

    if (requestCode == VIDEO_CAPTURE) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this, "Video saved to:\n" +
                    videoUri, Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            Toast.makeText(this, "Video recording cancelled.",
                    Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(this, "Failed to record video",
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

The above code example simply displays a toast message indicating the success of the recording intent session. In the event of a successful recording, the path to the stored video file is displayed.

When executed, the video capture intent (Figure 65-1) will launch and provide the user the opportunity to record video.

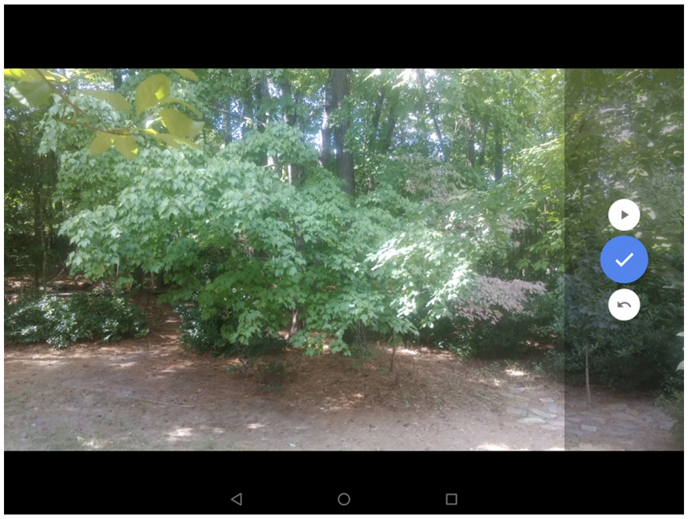

Figure 65-1

65.3 Calling the Image Capture Intent

In addition to the video capture intent, Android also includes an intent designed for taking still photos using the built-in camera, launched by referencing *MediaStore.ACTION_IMAGE_CAPTURE*:

```
private static final int IMAGE_CAPTURE = 102;

Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, IMAGE_CAPTURE);
```

As with video capture, the intent may be passed the location and file name into which the image is to be stored, or left to use the default location and naming convention.

65.4 Creating an Android Studio Video Recording Project

In the remainder of this chapter, a very simple application will be created to demonstrate the use of the video capture intent. The application will consist of a single button which will launch the video capture intent. Once video has been recorded and the video capture intent dismissed, the application will simply display the path to the video file as a Toast message. The VideoPlayer application created in the previous chapter may then be modified to play back the recorded video.

Create a new project in Android Studio, entering *CameraApp* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of an Empty Activity named *CameraAppActivity* with a layout file named *activity_camera_app.*

65.5 Designing the User Interface Layout

Navigate to *app* -> *res* -> *layout* and double-click on the *activity_camera_app.xml* layout file to load it into the Layout Editor tool.

With the Layout Editor tool in Design mode, delete the default "Hello World!" text view and replace it with a Button view positioned in the center of the layout canvas. Change the text on the button to read "Record Video" and extract the text to a string resource. Also, assign an *onClick* property to the button so that it calls a method named *startRecording* when selected by the user:

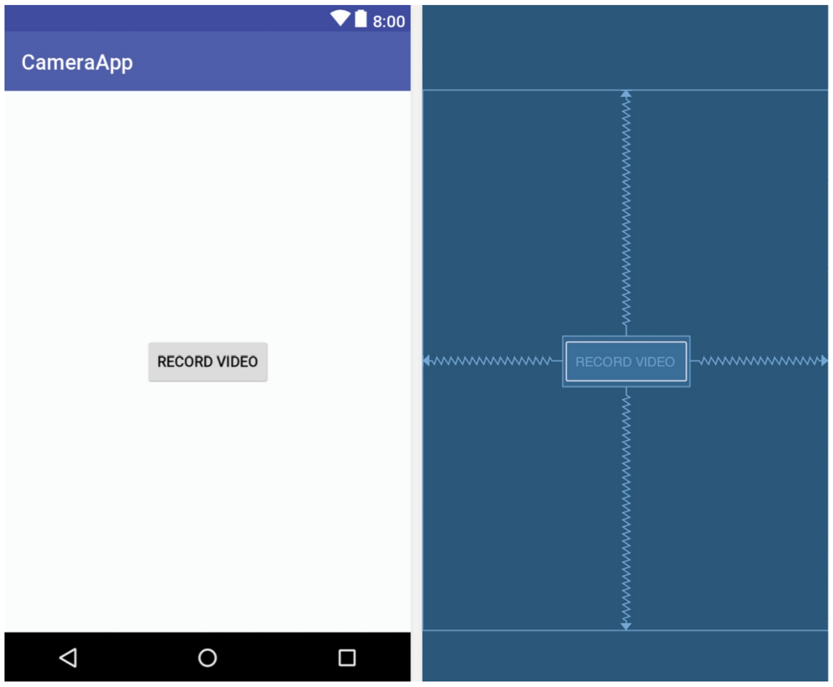

Figure 65-2

Remaining within the Attributes tool window, change the ID to *recordButton.*

65.6 Checking for the Camera

Before attempting to launch the video capture intent, the application first needs to verify that the device on which it is running actually has a camera. For the purposes of this example, we will simply make use of the previously outlined *hasCamera()* method, this time checking for any camera type. In the event that a camera is not present, the Record Video button will be disabled.

Edit the *CameraAppActivity.java* file and modify it as follows:

```
package com.ebookfrenzy.cameraapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.pm.PackageManager;
import android.widget.Button;

public class CameraAppActivity extends AppCompatActivity {

       @Override
       protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_camera_app);

              Button recordButton =
                (Button) findViewById(R.id.recordButton);

              if (!hasCamera())
                     recordButton.setEnabled(false);
       }

       private boolean hasCamera() {
            return (getPackageManager().hasSystemFeature(
                            PackageManager.FEATURE_CAMERA_ANY));
       }
}
```

65.7 Launching the Video Capture Intent

The objective is for the video capture intent to launch when the user selects the *Record Video* button. Since this is now configured to call a method named *startRecording()*, the next logical step is to implement this method within the *CameraAppActivity.java* source file:

```
package com.ebookfrenzy.cameraapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.pm.PackageManager;
import android.widget.Button;
import android.net.Uri;
import android.os.Environment;
import android.provider.MediaStore;
import android.content.Intent;
import android.view.View;

public class CameraAppActivity extends AppCompatActivity {

    private static final int VIDEO_CAPTURE = 101;

    public void startRecording(View view)
    {
        Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
        startActivityForResult(intent, VIDEO_CAPTURE);
    }
.
.
}
```

65.8 Handling the Intent Return

When control returns back from the intent to the application's main activity, the *onActivityResult()* method will be called. All that this method needs to do for this example is verify the success of the video capture and display the path of the file into which the video has been stored:

```
.
.
import android.widget.Toast;
.
.
public class CameraAppActivity extends AppCompatActivity {
.
.
    protected void onActivityResult(int requestCode,
              int resultCode, Intent data) {

        Uri videoUri = data.getData();

        if (requestCode == VIDEO_CAPTURE) {
            if (resultCode == RESULT_OK) {
                Toast.makeText(this, "Video saved to:\n" +
                        videoUri, Toast.LENGTH_LONG).show();
            } else if (resultCode == RESULT_CANCELED) {
                Toast.makeText(this, "Video recording cancelled.",
                        Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(this, "Failed to record video",
                        Toast.LENGTH_LONG).show();
            }
    }
.
.
}
```

65.9 Testing the Application

Compile and run the application on a physical Android device or emulator session, touch the record button and use the video capture intent to record some video. Once completed, stop the video recording. Play back the recording by selecting the play button on the screen. Finally, touch the *Done* (sometimes represented by a check mark) button on the screen to return to the CameraApp application. On returning, a Toast message should appear stating that the video has been stored in a specific location on the device (the exact location will differ from one device type to another) from where it can be moved, stored or played back depending on the requirements of the app.

65.10 Summary

Most Android tablet and smartphone devices include a camera that can be accessed by applications. While there are a number of different approaches to adding camera support to applications, the Android video and image capture intents provide a simple and easy solution to capturing video and images.