MULTIMEDIA

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

MULTIMEDIA MATERIALS

Building Apps with Multimedia

https://developer.android.com/training/building-multimedia.html

Also see Media Playback

https://developer.android.com/guide/topics/media/mediaplayer.html

AUDIO

Android maintains a separate audio stream for playing music, alarms, notifications, the incoming call ringer, system sounds, in-call volume, and DTMF tones.

This allows users to control the volume of each stream independently.

Most streams are restricted to system events.

Unless your app is a replacement alarm clock, you'll almost certainly be playing your audio using the **STREAM_MUSIC** stream.

By default, pressing the volume controls modifies the volume of the active audio stream.
If your app isn't currently playing anything, hitting the volume keys adjusts the ringer volume.

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

AUDIO HARDWARE CONTROLS

Media playback buttons such as play, pause, stop, skip, and previous are available on some handsets and many connected or wireless headsets.

To respond to media button clicks, you need to register a BroadcastReceiver in your manifest that listens for systems broadcasts.

```
<receiver android:name=".RemoteControlReceiver">
    <intent-filter>
        <action android:name="android.intent.action.MEDIA_BUTTON" />
    </intent-filter>
</receiver>
```

```
public class RemoteControlReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (Intent.ACTION_MEDIA_BUTTON.equals(intent.getAction())) {
            KeyEvent event = (KeyEvent)intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
            if (KeyEvent.KEYCODE_MEDIA_PLAY == event.getKeyCode()) {
                // Handle key press.
            }
        }
    }
```

Don’t forget to register and unregister any Broadcast Receivers

AUDIO FOCUS

With multiple apps potentially playing audio it's important to think about how they should interact. To avoid every music app playing at the same time, Android uses audio focus to moderate audio playback—only apps that hold the audio focus should play audio.

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
...

// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
                                  // Use the music stream.
                                  AudioManager.STREAM_MUSIC,
                                  // Request permanent focus.
                                  AudioManager.AUDIOFOCUS_GAIN);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    am.registerMediaButtonEventReceiver(RemoteControlReceiver);
    // Start playback.
}
```

```
// Abandon audio focus when playback complete
am.abandonAudioFocus(afChangeListener);
```

AUDIO LISTENER

```
OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT
            // Pause playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
            // Resume playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_LOSS) {
            am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
            am.abandonAudioFocus(afChangeListener);
            // Stop playback
        }
    }
};
```

OUTPUT HARDWARE

Most devices have a built-in speaker, headphone jacks for wired headsets, and many also feature Bluetooth connectivity and support for A2DP audio.

You can query the **AudioManager** to determine where audio is routed.

The system broadcasts an Intent when the output device changes. E.g. ACTION_AUDIO_BECOMING_NOISY intent when headset unplugged.

```
if (isBluetoothA2dpOn()) {
    // Adjust output for Bluetooth.
} else if (isSpeakerphoneOn()) {
    // Adjust output for Speakerphone.
} else if (isWiredHeadsetOn()) {
    // Adjust output for headsets
} else {
    // If audio plays and noone can hear it, is it still playing?
}
```

TAKING A PHOTO

The simplest way is to use an inbuilt camera app.

1. Request permission

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera"
                  android:required="true" />
    ...
</manifest>
```

2. Use an Explicit Intent

```
static final int REQUEST_IMAGE_CAPTURE = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}
```

SHOW THE IMAGE

As a thumbnail:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        mImageView.setImageBitmap(imageBitmap);
    }
}
```

This thumbnail image from "data" might be good for an icon, but not a lot more. Dealing with a full-sized image takes a bit more work.

THE FULL-SIZE PHOTO

Generally, any photos that the user captures with the device camera should be saved on the device in the public external storage so they are accessible by all apps.

The proper directory for shared photos is provided by **getExternalStoragePublicDirectory()**, with the **DIRECTORY_PICTURES** argument.

Because the directory provided by this method is shared among all apps, reading and writing to it requires the READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE permissions, respectively.

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

COLLISION-RESISTANT FILE NAMES

```
String mCurrentPhotoPath;

private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES);
    File image = File.createTempFile(
        imageFileName,  /* prefix */
        ".jpg",         /* suffix */
        storageDir      /* directory */
    );

    // Save a file: path for use with ACTION_VIEW intents
    mCurrentPhotoPath = "file:" + image.getAbsolutePath();
    return image;
}
```

SAVE THE PHOTO

```
static final int REQUEST_TAKE_PHOTO = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                    Uri.fromFile(photoFile));
            startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO);
        }
    }
}
```

SAVE THE PHOTO

```
static final int REQUEST_TAKE_PHOTO = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
                                                      "com.example.android.fileprovider",
                                                      photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO);
        }
    }
}
```

SAVE THE PHOTO

```
<application>
    ...
    <provider
         android:name="android.support.v4.content.FileProvider"
         android:authorities="com.example.android.fileprovider"
         android:exported="false"
         android:grantUriPermissions="true">
         <meta-data
             android:name="android.support.FILE_PROVIDER_PATHS"
             android:resource="@xml/file_paths"></meta-data>
     </provider>
     ...
</application>
```

For more recent apps targeting Android 7.0 (API level 24) and higher, passing a file:// URI across a package boundary causes a FileUriExposedException. Therefore, it is better to used the generic way of storing files using a FileProvider.

SHARE THE PHOTO

The easiest way to make your photo accessible is to make it accessible from the system's Media Provider.

The following demonstrates how to invoke the system's media scanner to add your photo to the Media Provider's database, making it available in the Android Gallery application and to other apps.

```
private void galleryAddPic() {
    Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
    File f = new File(mCurrentPhotoPath);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    this.sendBroadcast(mediaScanIntent);
}
```

Note: this will not work if you saved to a private directory: e.g. if you used the call getExternalFilesDir().

RECORDING VIDEO

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera"
                  android:required="true" />
    ...
</manifest>
```

```
static final int REQUEST_VIDEO_CAPTURE = 1;

private void dispatchTakeVideoIntent() {
    Intent takeVideoIntent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    if (takeVideoIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takeVideoIntent, REQUEST_VIDEO_CAPTURE);
    }
}
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_VIDEO_CAPTURE && resultCode == RESULT_OK) {
        Uri videoUri = intent.getData();
        mVideoView.setVideoURI(videoUri);
    }
}
```

CONTROL THE CAMERA

If you want to directly incorporate camera functionality into your app you will need to use the **Camera** api:

https://developer.android.com/reference/android/hardware/Camera.html

See also, building a camera app:

https://developer.android.com/guide/topics/media/camera.html

Or you can use the **Camera2** api:

https://developer.android.com/reference/android/hardware/camera2/package-summary

Video intro:

https://www.youtube.com/watch?v=Bi4QjMfSOE0

CAMERA PREVIEW

The preview requires an implementation of the **android.view.SurfaceHolder.Callback** interface, which is used to pass image data from the camera hardware to the application.

```
class Preview extends ViewGroup implements SurfaceHolder.Callback {

    SurfaceView mSurfaceView;
    SurfaceHolder mHolder;

    Preview(Context context) {
        super(context);

        mSurfaceView = new SurfaceView(context);
        addView(mSurfaceView);

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

START THE PREVIEW

```
public void setCamera(Camera camera) {
    if (mCamera == camera) { return; }

    stopPreviewAndFreeCamera();

    mCamera = camera;

    if (mCamera != null) {
        List<Size> localSizes = mCamera.getParameters().getSupportedPreviewSizes();
        mSupportedPreviewSizes = localSizes;
        requestLayout();

        try {
            mCamera.setPreviewDisplay(mHolder);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Important: Call startPreview() to start updating the preview
        // surface. Preview must be started before you can take a picture.
        mCamera.startPreview();
    }
}
```

RELEASE THE CAMERA

```
public void surfaceDestroyed(SurfaceHolder holder) {
    // Surface will be destroyed when we return, so stop the preview.
    if (mCamera != null) {
        // Call stopPreview() to stop updating the preview surface.
        mCamera.stopPreview();
    }
}

/**
 * When this function returns, mCamera will be null.
 */
private void stopPreviewAndFreeCamera() {

    if (mCamera != null) {
        // Call stopPreview() to stop updating the preview surface.
        mCamera.stopPreview();

        // Important: Call release() to release the camera for use by other
        // applications. Applications should release the camera immediately
        // during onPause() and re-open() it during onResume()).
        mCamera.release();

        mCamera = null;
    }
}
```

DRAWING EXAMPLES

See examples on Moodle

Targets.zip

BouncingBall.zip

GRAPHICS MATERIALS

Building Apps with Graphics & Animation

https://developer.android.com/training/building-graphics.html

This tutorial provides lots of details on:

- Dealing with bitmap images effectively
- Displaying graphics with OpenGL ES
- Animation and transition for Views
- Creating and adding animations

VIDEOS

DevBytes: Bitmap Allocation

https://www.youtube.com/watch?v=rsQet4nBVi8

DevBytes: Making Apps Beautiful

https://www.youtube.com/watch?v=pMRnGDR6Cu0

DevBytes: Transitions

https://www.youtube.com/watch?v=S3H7nJ4QaD8

More generally, for lots of excellent videos on Android development, see:
https://www.youtube.com/user/androiddevelopers/playlists

QUESTIONS?

Contact:

d.coyle@ucd.ie