Unit – 4 **Multimedia and System Services**

Book – 1 Professional Android 2 Application Development, by Reto Meier, Wrox Publication

Book – 2 Beginning Android Programming with Android Studio – J. F. DiMarzio (4th Edition)

Playing Audio and Video

- Latest versions of Android support the following multimedia formats for playback as part of the base framework.
- Note that some devices may support playback of additional file formats:
- **Audio**
  - .mp3, .mkv, .wav, .mid, .xmf, .mxmf, RTTTL/RTX (.rtttl, .rtx), OTA (.ota), iMelody (.imy) , .ogg, .3gp
- **Video**
  - MPEG-4 (.mp4), .webm, MPEG-4 SP (.3gp), .mkv
- **Images**
  - BMP, JPEG, GIF, PNG, WEBP, HEIF

Media Player

- Multimedia playback in Android is handled by the “MediaPlayer” class.
- You can play media stored in application resources, local files, Content Providers, or streamed from a network URL.
- In each case, the file format and type of multimedia being played is abstracted from you as a developer.
- The Media Player’s management of audio and video files and streams is handled as a state machine.
- In simple terms, transitions through the state machine can be described as follows:
  - Initialize the Media Player with media to play.
  - Prepare the Media Player for playback.
  - Start the playback.
  - Pause or stop the playback prior to its completing.
  - Playback complete.

Initialize the Media Player

- To play a media resource you need to create a new “MediaPlayer” instance, initialize it with a media source, and prepare it for playback.

  Context appContext = getApplicationContext();

  MediaPlayer resourcePlayer = MediaPlayer.create(appContext,

  R.raw.my_audio);

Preparing Audio for Playback

- There are a number of ways you can play audio content through the Media Player.

1. You can include it as an application resource
   - You can include audio files in your application package by adding them to the res/raw folder of your resources hierarchy.
2. Play it from local files or Content Providers
3. Stream it from a remote URL

Initializing Audio Content for Playback

- To playback audio content using the Media Player, you need to create a new Media Player object and set the data source of the audio in question.

```
Context appContext = getApplicationContext();
MediaPlayer resourcePlayer = MediaPlayer.create(appContext,
    R.raw.my_audio); //through resource identifier
MediaPlayer filePlayer = MediaPlayer.create(appContext,
    Uri.parse("file:///sdcard/localfile.mp3"));//from local file
MediaPlayer urlPlayer = MediaPlayer.create(appContext,
    Uri.parse("http://site.com/audio/audio.mp3")); //online
    audio
MediaPlayer contentPlayer = MediaPlayer.create(appContext,
    Settings.System.DEFAULT_RINGTONE_URI); //local content
    provider
```

Preparing for Video Playback

- Playback of video content is slightly more involved than audio. To show a video, you must specify a display surface on which to show it.
- There are two alternatives for the playback of video content.
- The first, using the Video View control, encapsulates the creation of a display surface and allocation and preparation of video content within a Media Player.
- The second technique allows you to specify your own display surface and manipulate the underlying Media Player instance directly.

Playing Video Using the Video View

- The simplest way to play back video is to use the ***VideoView*** control.
- The Video View includes a Surface on which the video is displayed and encapsulates and manages a Media Player to manage the video playback.
- The Video View supports the playback of local or streaming video as supported by the Media Player component.
- Video Views conveniently encapsulate the initialization of the Media Player.
- To assign a video to play, simply call ***setVideoPath*** or ***setVideoUri*** to specify the path to a local file, or the URI of a Content Provider or remote video stream:

```
streamingVideoView.setVideoUri("http://www.mysite.com/videos/my
    video.3gp");
localVideoView.setVideoPath("/sdcard/test2.3gp");
```

Playing Video Using the Video View

- Once initialized, you can control playback using the start, stopPlayback, pause, and seekTo methods.
- The Video View also includes the ***setKeepScreenOn*** method to apply a screen Wake Lock that will prevent the screen from being dimmed while playback is in progress.
- Following is the simple skeleton code used to assign a video to a Video View and control playback

```
VideoView videoView = (VideoView)findViewById(R.id.surface);
videoView.setKeepScreenOn(true);
videoView.setVideoPath("/sdcard/test2.3gp");
videoView.start();
[ . . . do something . . . ]
videoView.stopPlayback();
```

Accessing Camera

- The easiest way to take a picture using the device camera is using the ACTION_IMAGE_CAPTURE Media Store static constant in an Intent passed to ***startActivityForResult***.

  startActivityForResult(new Intent(MediaStore.ACTION_IMAGE_CAPTURE),TAKE_PICTURE);

- This will launch the camera Activity, allowing users to modify the image settings manually, and preventing you from having to rewrite the entire camera application.

Capturing Image

- The image capture action supports two modes, thumbnail and full image.

1. **Thumbnail** By default, the picture taken by the image capture action will return a thumbnail Bitmap in the "data extra" within the Intent parameter returned in ***onActivityResult***.
2. **Full image** If you specify an output URI using a "MediaStore.EXTRA_OUTPUT extra" in the launch Intent, the full-size image taken by the camera will be saved to the specified location.
   - In this case no thumbnail will be returned in the Activity result callback and the result Intent data will be null.

Capturing Thumbnail

```
private static int TAKE_PICTURE = 1;
    private void getThumbailPicture() {
    Intent intent = new
            Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    startActivityForResult(intent, TAKE_PICTURE);
    }
```

Capturing Full Image

```
private Uri outputFileUri;
private void saveFullImage() {
    Intent intent = new
          Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    File file = new
          File(Environment.getExternalStorageDirectory(),
                    "test.jpg");
    outputFileUri = Uri.fromFile(file);
intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
startActivityForResult(intent, TAKE_PICTURE);
}
```

Accessing Camera Hardware Directly

- To access the camera hardware directly, you need to add the CAMERA permission to your application manifest.

  <uses-permission android:name="android.permission.CAMERA"/>
- Use the Camera class to adjust camera settings, specify image preferences, and take pictures.
- To access the Camera Service, use the static open method on the Camera class.

  Camera camera = Camera.open();
- When your application has finished with the Camera, remember to relinquish your hold on it by calling release.

  camera.release();

Controlling and Monitoring Camera Settings

- The camera settings are stored using a ***Camera.Parameters*** object, accessible by calling the ***getParameters*** method on the Camera object.

  Camera.Parameters parameters = camera.getParameters();

- In order to modify the camera settings, use the set methods on the Parameters object before calling the Camera's ***setParameters*** method and passing in the modified Parameters object.

  camera.setParameters(parameters);

Controlling and Monitoring Camera Settings

- **Following are some popular camera parameters**
- **[get/set]SceneMode** Takes or returns a SCENE_MODE_* static string constant from the Camera Parameters class. Each scene mode describes a particular scene type (party, beach, sunset,etc.).
- **[get/set]FlashMode** Takes or returns a FLASH_MODE_* static string constant. Lets you specify the flash mode as on, off, red-eye reduction, or flashlight mode.
- **[get/set]WhiteBalance** Takes or returns a WHITE_BALANCE_* static string constant to describe the white balance of the scene being photographed.
- **[get/set]ColorEffect** Takes or returns a EFFECT_* static string constant to modify how the image is presented. Available color effects include sepia tone or black and white.
- **[get/set]FocusMode** Takes or returns a FOCUS_MODE_* static string constant to specify how the camera autofocus should attempt to focus the camera.

Using Text Messages (SMS)

- Android comes with a built-in SMS application that enables you to send and receive SMS messages.
- However, in some cases you might want to integrate SMS capabilities into your own Android application.
- The key features of messaging are send and receive SMS messages in your Android applications.

Sending SMS Messages Programmatically

- Android uses a permissions-based policy whereby all the permissions needed by an application must be specified in the AndroidManifest.xml file.
- This ensures that when the application is installed, the user knows exactly which access permissions it requires.

  <uses-permission android:name=”android.permission.SEND_SMS”></uses-permission>
- Because sending SMS messages incurs additional costs on the user’s end, indicating the SMS permissions in the AndroidManifest.xml file enables users to decide whether to allow the application to install or not.

Sending SMS Messages Programmatically

- To send an SMS message programmatically, you use the ***SmsManager*** class.
- Unlike other classes, you do not directly instantiate this class; instead, you call the ***getDefault()*** static method to obtain a SmsManager object.
- You then send the SMS message using the ***sendTextMessage()*** method:

```
private void sendSMS(String phoneNumber, String message)
{
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber, null, message, sentPI,
    deliveredPI);
}
```

Sending SMS Messages Programmatically

- Following are the five arguments to the sendTextMessage() method:

1. destinationAddress — Phone number of the recipient
2. scAddress — Service centre address; use null for default SMSC
3. text — Content of the SMS message
4. sentIntent — Pending intent to invoke when the message is sent (To indicate SMS is sent)
5. deliveryIntent — Pending intent to invoke when the message has been delivered (To indicate SMS is delivered)

Sending SMS Messages Using Intents

- Using the SmsManager class, you can send SMS messages from within your application without the need to involve the built-in Messaging application.
- However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it do all the work of sending the message.
- To activate the built-in Messaging application from within your application, you can use an Intent object together with the MIME type ***"vnd.android-dir/mms-sms"***
- You can send your SMS to multiple recipients by simply separating each phone number with a semicolon in the ***putExtra()*** method.

Sending SMS Messages Using Intents

```
public void onClick(View v)
{
    //send SMS(“5556”, “Hello my friends!”);
    Intent i = new Intent(android.content.Intent.ACTION_VIEW);
    i.putExtra(“address”, “5556; 5558; 5560”);
    i.putExtra(“sms_body”, “Hello my friends!”);
    i.setType(“vnd.android-dir/mms-sms”);
    startActivity(i);
}
```

- If you use this method to invoke the Messaging application, there is no need to ask for the SMS_SEND permission in AndroidManifest.xml because your application is ultimately not the one sending the message.

Receiving SMS

- Besides sending SMS messages from your Android applications, you can also receive incoming SMS messages from within your application by using a ***BroadcastReceiver*** object.
- This is useful when you want your application to perform an action when a certain SMS message is received.
- For example, you might want to track the location of your phone in case it is lost or stolen.
- In this case, you can write an application that automatically listens for SMS messages containing some secret code.
- Once that message is received, you can then send an SMS message containing the location’s coordinates back to the sender.

Receiving SMS

- Following statements to be added in AndroidMenifest.xml file.

```
<receiver android:name=”.SMSReceiver”>
   <intent-filter>
      <action android:name=
      “android.provider.Telephony.SMS_RECEIVED” />
   </intent-filter>
</receiver>
<uses-permission
  android:name=”android.permission.RECEIVE_SMS”>
</uses-permission>
```

Receiving SMS

- To listen for incoming SMS messages, you create a ***BroadcastReceiver*** class.
- The BroadcastReceiver class enables your application to receive intents sent by other applications using the ***sendBroadcast()*** method.
- Essentially, it enables your application to handle events raised by other applications.
- When an intent is received, the ***onReceive()*** method is called; hence, you need to override this.

Introduction to Geolocation

- One category of apps that is very popular is location-based services, commonly known as LBS.
- LBS apps track your location, and may offer additional services such as locating amenities nearby, as well as offering suggestions for route planning, and so on.
- Of course, one of the key ingredients in a LBS app is maps, which present a visual representation of your location.
- Google Maps is one of the many applications bundled with the Android platform.
- In addition to simply using the Maps application, you can also embed it into your own applications.

Creating a Project

- To start, you need to first create an Android project to display Google Maps in activity

1. Using Android Studio, create an Android project and name it, say LBS.
2. From the Create New Project Wizard, select Google Maps Activity.

Obtaining the Maps API Key

- Beginning with the Android SDK release v1.0, you need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application.
- When you apply for the key, you must also agree to Google’s terms of use, so be sure to read them carefully.

1. To get a Google Maps key, open the ***google_maps_api.xml*** file that was created in your project.
2. Within this file is link to create a new Google Maps key. Simply copy and paste the link into your browser and follow the instructions.
3. Save the key that Google gives you as it will be needed later in this project.

Displaying Google Maps

- To display Google Maps in your application, you first need to ACCESS_FINE_LOCATION permission in your manifest file.

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

- This is created for you automatically when you selected to set up a Google Maps Activity.
- In order to test application on the Android emulator, be sure to create an Emulator with SDK version that includes Google Play Service as the selected target.

Common Errors in Displaying Maps

- If instead of seeing Google Maps displayed you see an empty screen with grids, then:

1. Most likely you are using the wrong API key in the google_maps_api.xml file.
2. You may not having internet connection access to you emulator/device.
3. Package name that you supplied with you registered for your key is not matching the package name in your application

Animations in Android

- Android supports three types of animations:
- **Property Animation:** Introduced in Android 3.0 (API level 11), the property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well
- **Drawable Animation:** Drawable animation involves displaying Drawable resources one after another, like a roll of film. This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.

Animations in Android

- **View Animation:** View Animation is the older system and can only be used for Views. It is relatively easy to setup and offers enough capabilities to meet many application's needs. There are two types of animations that can be created using this method
  - **Tween Animation:** Creates an animation by performing a series of transformations on a single image with an Animation
  - **Frame Animation:** creates an animation by showing a sequence of frames (images) in order with an AnimationDrawable

System Services

- A Service is an application component that can perform long-running operations in the background and does not provide a user interface.
- Another application component can start a service and it will continue to run in the background even if the user switches to another application.
- Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC).
- For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

System Services

- A service may be used in two ways:

1. A service is "started" when an application component (such as an activity) starts it by calling *startService()*.

- Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
- Usually, a started service performs a single operation and does not return a result to the caller.
- For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

System Services

2. A service is "bound" when an application component binds to it by calling *bindService()*.

- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across process boundaries with inter-process communication (IPC).
- A bound service runs only as long as another application component is bound to it.
- Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

System Services

- The same service can work both ways
  - it can be started (to run indefinitely) and also allow binding.
- It is simply a matter of which of the callback methods are implemented:
- Implement *onStartCommand()* to allow components to start it, *onBind()* to allow binding and both to allow both ways of working.

System Services

- The Android system will force-stop a service when memory is low and it must recover system resources for the activity that has user focus.
- If the service is bound to an activity that has user focus, then it is less likely to be killed.
- If the system kills your service, it restarts it as soon as resources become available again.