Module 3.1

Accessing Android Hardware

Objectives:

At the end of the module the students should be able to:

- Take pictures and control the camera
- Playing audio and video with the `MediaPlayer`
- Recording Audio using `MediaRecorder`

Android's application-neutral APIs provide low-level access to the increasingly diverse hardware commonly available on mobile devices. The ability to monitor and control these hardware features provides a great incentive for application development on the Android platform.

The hardware APIs available include:

- A telephony package that provides access to calls and phone status.
- A multimedia playback and recording library.
- Access to the device camera for taking pictures and previewing video.
- Extensible support for sensor hardware.
- Accelerometer and compass APIs to monitor orientation and movement.
- Communications libraries for managing Bluetooth, network, and Wi-Fi hardware.

Using Intents to use Existing Camera Apps

You will use `MediaStore.ACTION_IMAGE_CAPTURE` to launch an existing camera application installed on your phone.

```
Intent intent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE)
```

Apart from the above, there are other available Intents provided by `MediaStore`. They are listed as follows

No.	Intent type and description
1	`ACTION_IMAGE_CAPTURE_SECURE` It returns the image captured from the camera , when the device is secured
2	`ACTION_VIDEO_CAPTURE` It calls the existing video application in android to capture video
3	`EXTRA_SCREEN_ORIENTATION` It is used to set the orientation of the screen to vertical or landscape
4	`EXTRA_FULL_SCREEN` It is used to control the user interface of the `ViewImage`
5	`INTENT_ACTION_VIDEO_CAMERA` This intent is used to launch the camera in the video mode
6	`EXTRA_SIZE_LIMIT` It is used to specify the size limit of video or image capture size

Now you will use the function `startActivityForResult()`to launch this activity and wait for its result. Its syntax is given below:

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

This launches a Camera application to take the photo, providing your users with the full suite of camera functionality without you having to rewrite the native Camera application.

Once users are satisfied with the image, the result is returned to your application within the Intent received by the `onActivityResult()`handler. When the user finishes taking a picture or video (or cancels the operation), the system calls this method.

By default, the picture taken will be returned as a thumbnail, available as a raw bitmap within the data extra within the returned Intent. To obtain a full image, you must specify a target fi le in which to store it, encoded as a URI passed in using the `MediaStore.EXTRA_OUTPUT` extra in the launch Intent

```
// Create an output file to external directory.
File file = new File(Environment.getExternalStorageDirectory(), “sample.jpg”);
Uri outputFileUri = Uri.fromFile(file);
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
startActivityForResult(intent, TAKE_PICTURE);
```

Displaying receive output to ImageView

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, TAKE_PICTURE);
@Override
protected void onActivityResult(intrequestCode, intresultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);
Bitmap bp = (Bitmap) data.getExtras().get("data");
imgView.setImageBitmap(bp);
}
```

Using Camera API directly in our Application

Before using the Camera API, you should make sure your manifest has the appropriate declarations to allow use of camera hardware and other related features.

Camera Permission - Your application must request permission to use a device camera

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

Note: No need to request permission when using Camera via intent.

Camera Features - Your application must declare use of camera features

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

Detecting Camera Hardware

If your application does not specifically require a camera using a manifest declaration, you should check to see if a camera is available at runtime. To perform this check, use the `PackageManager.hasSystemFeature()` method, as shown in the example code below:

```
private boolean checkCameraHardware(Context context) {
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        // this device has a camera
        return true;
    } else {
        // no camera on this device
        return false;
    }
}
```

Accessing cameras

To access the primary camera, use the `Camera.open()` method and be sure to catch any exceptions, as shown in the code below:

```
public static Camera getCameraInstance(){
    Camera c = null;
    try {
        c = Camera.open(); // attempt to get a Camera instance
    }
    catch (Exception e){
        // Camera is not available (in use or does not exist)
    }
    return c; // returns null if camera is unavailable
}
```

Creating a preview class

For users to effectively take pictures or video, they must be able to see what the device camera sees. A camera preview class is a `SurfaceView` that can display the live image data coming from a camera, so users can frame and capture a picture or video.

The following example code demonstrates how to create a basic camera preview class that can be included in a`View` layout. This class implements `SurfaceHolder.Callback` in order to capture the callback events for creating and destroying the view, which are needed for assigning the camera preview input.

```
public class ShowCamera extends SurfaceView implements SurfaceHolder.Callback {
private SurfaceHolder holdMe;
private Camera theCamera;
publicShowCamera(Context context,Camera camera) {
          super(context);
          theCamera = camera;
          holdMe = getHolder();
          holdMe.addCallback(this);
   }
   @Override
   public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
   }
   @Override
public void surfaceCreated(SurfaceHolder holder) {
try   {
          theCamera.setPreviewDisplay(holder);
          theCamera.startPreview();
      } catch (IOException e) {
      }
   }
   @Override
public void surfaceDestroyed(SurfaceHolder arg0) {
}
}
```

Placing preview in a layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal" >
          <FrameLayout
          android:id="@+id/camera_preview"
          android:layout_width="fill_parent"
          android:layout_height="199dp"
          android:layout_weight="0.96" />

          <Button
          android:id="@+id/button_capture"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:onClick="snapIt"
          android:text="@string/Capture" />
</LinearLayout>
```

In the activity for your camera view, add your preview class to the `FrameLayout` element . Your camera activity must also ensure that it releases the camera when it is paused or shut down. The following example shows how to modify a camera activity to attach the preview.

```
public class MainActivity extends Activity {
Camera cameraObject;
private CameraPreview mPreview;

          protected void onCreate(Bundle savedInstanceState) {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.activity_main);
                    cameraObject = getCameraInstance();
                    mPreview = new ShowCamera(this, cameraObject);
                    FrameLayout fra = (FrameLayout)findViewById(R.id.camera_preview);
                    fra.addView(mPreview);
          }
}
```

Playing Audio and Video with the Media Player

Android 4.0.3 (API level 15) supports the following multimedia formats for playback as part of the base framework. Note that some devices may support playback of additional fi le formats:

Audio

- AAC LC/LTP
- HE-AACv1 (AAC+)
- HE-AACv2 (Enhanced AAC+)
- AMR-NB
- AMR-WB
- MP3
- MIDI
- OggVorbis

- PCM/WAVE
- FLAC (on devices running Android 3.1 or above)

Image
- JPEG
- PNG
- WEBP (on devices running Android 4.0 or above)
- GIF
- BMP

Video
- H.263
- H.264 AVC
- MPEG-4 SP
- VP8 (on devices running Android 2.3.3 or above)

Playing Audio and Video

The playback of audio and video within Android applications is handled primarily through the `MediaPlayer` class. Using the Media Player, you can play media stored in application resources, local files, Content Providers, or streamed from a network URL. The Media Player's management of audio and video files and streams is handled as a state machine. In the most simplistic terms, transitions through the state machine can be described as follows:

1. Initialize the Media Player with media to play.
2. Prepare the Media Player for playback.
3. Start the playback.
4. Pause or stop the playback prior to its completing.
5. The playback is complete.

To play a media resource, you need to create a new `MediaPlayer` instance, initialize it with a media source, and prepare it for playback. The following code describes how to initialize and prepare the Media Player. After that, you'll learn to control the playback to start, pause, stop, or seek the prepared media.

To stream Internet media using the Media Player, your application must include the INTERNET permission:

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

Note:

Android supports a limited number of simultaneous Media Player objects; not releasing them can cause runtime exceptions when the system runs out. When you finish playback, call release on your Media Player object to free the associated resources:

```
mediaPlayer.release();
```

Preparing Audio for Playback

There are a number of ways you can play audio content through the Media Player. You can include it as an application resource, play it from local fi les or Content Providers, or stream it from a remote URL.

To include audio content as an application resource, add it to the res/raw folder of your resources hierarchy. Raw resources are not compressed or manipulated in any way when packaged into your application, making them an ideal way to store pre-compressed fi les such as audio fi les.

Initializing Audio Content for Playback

To play back audio content using a Media Player, you need to create a new Media Player object and set the data source of the audio in question. You can do this by using the static create method, passing in the Activity Context and any one of the following audio sources:

- A resource identifi er (typically for an audio fi le stored in the res/raw resource folder)
- A URI to a local fi le (using the file:// schema)

- A URI to an online audio resource (as a URL)
- A URI to a row within a Content Provider that returns an audio fi le

```
// Load an audio resource from a package resource.
MediaPlayer resourcePlayer =
MediaPlayer.create(this, R.raw.my_audio);
// Load an audio resource from a local file.
MediaPlayer filePlayer = MediaPlayer.create(this,
Uri.parse(“file:///sdcard/localfile.mp3”));
// Load an audio resource from an online resource.
MediaPlayer urlPlayer = MediaPlayer.create(this,
Uri.parse(“http://site.com/audio/audio.mp3”));
// Load an audio resource from a Content Provider.
MediaPlayer contentPlayer = MediaPlayer.create(this,
Settings.System.DEFAULT_RINGTONE_URI);
```

Alternatively, you can use the `setDataSource` method on an existing Media Player instance. This method accepts a fi le path, Content Provider URI, streaming media URL path, or File Descriptor. When using the `setDataSource` method, it is vital that you call prepare on the Media Player before you begin playback.

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(“/sdcard/noypi.mp3”);
mediaPlayer.prepare();
```

Preparing Video for Playback

Playback of video content is slightly more involved than audio. To play a video, you first must have a Surface on which to show it. There are two alternatives for the playback of video content. The first technique, using the `VideoView` class, encapsulates the creation of a Surface and allocation and preparation of video content using a Media Player. The second technique allows you to specify your own 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 Video View. The Video View includes a Surface on which the video is displayed and encapsulates and manages a Media Player instance that handles the playback. After placing the Video View within the UI, get a reference to it within your code. You can then assign a video to play by calling its `setVideoPath` or `setVideoURI` methods to specify the path to a local file, or the URI of either a Content Provider or remote video stream:

```
final VideoView videoView = (VideoView)findViewById(R.id.videoView);
// Assign a local file to play
videoView.setVideoPath(“/sdcard/mycatvideo.3gp”);
// Assign a URL of a remote video stream
videoView.setVideoUri(myAwesomeStreamingSource);
```

When the video is initialized, you can control its 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 without requiring a special permission.

Below is the skeleton code used to assign a video to a Video View. It uses a Media Controller to control playback.

```
// Get a reference to the Video View.
final VideoView videoView = (VideoView)findViewById(R.id.videoView);
// Configure the video view and assign a source video.
videoView.setKeepScreenOn(true);
videoView.setVideoPath("/sdcard/thevoicekids.3gp");
// Attach a Media Controller
MediaController mediaController = new MediaController(this);
videoView.setMediaController(mediaController);
```

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below:

No.	Method and description
1	isPlaying() This method just returns true/false indicating the song is playing or not
2	seekTo(position) This method takes an integer, and move song to that particular second
3	getCurrentDuration() This method returns the current position of song in milliseconds
4	getDuration() This method returns the total time duration of song in milliseconds
5	reset() This method resets the media player
6	release() This method releases any resource attached with MediaPlayer object
7	setVolume(float leftVolume, float rightVolume) This method sets the up down volume for this player
8	setDataSource(FileDescriptor fd) This method sets the data source of audio/video file
9	selectTrack(int index) This method takes an integer, and select the track from the list on that particular index
10	getTrackInfo() This method returns an array of track information

Recording Audio using MediaRecorder

Android has a built in microphone through which you can capture audio and store it, or play it in your phone. There are many ways to do that but the most common way is through `MediaRecorder`class.

As usual, you have to state that app wants to record audio in the manifest file. Add these lines

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

Android provides `MediaRecorder` class to record audio or video. In order to use `MediaRecorder` class, you will first create an instance of `MediaRecorder` class. Its syntax is given below.

```
MediaRecorder recorder = new MediaRecorder();
```

Now you will set the source , output and encoding format and output file. Their syntax is given below.

```
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
recorder.setOutputFile(outputFile);
```

After specifying the audio source and format and its output file, we can then call the two basic methods `prepare` and `start` to start recording the audio.

```
recorder.prepare();
recorder.start();
```

Apart from these methods , there are other methods listed in the `MediaRecorder`class that allows you more control over audio and video recording.

No.	Method and description
1	setAudioSource() This method specifies the source of audio to be recorded
2	setVideoSource() This method specifies the source of video to be recorded
3	setOutputFormat() This method specifies the audio format in which audio to be stored
4	setAudioEncoder() This method specifies the audio encoder to be used
5	setOutputFile() This method configures the path to the file into which the recorded audio is to be stored
6	stop() This method stops the recording process.
7	release() This method should be called when the recorder instance is needed.