LabW6 – Media Access

Objectives:

1. Understand the multimedia components for Android SDK

Tasks:

1. Set up permissions for accessing different components in Android devices.
2. Use a basic layout.
3. Take a photo using the phone’s built-in Camera app.
4. Select a photo from the gallery.
5. Select a video from the gallery.
6. Record a video.
7. Handle the media files.
8. Audio recording
9. Implement your own camera interface.

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the file system, or from a data stream arriving over a network connection.

This tutorial shows you how to write a media-playing application that interacts with the user and the system to obtain good performance and a pleasant user experience.

Task 1: Setup permissions for accessing different components in Android devices

Beginning in **Android 6.0 (API level 23),** users grant permissions to apps while the app is running, not when they install the app. This approach streamlines the app install process, since the user does not need to grant permissions when they install or update the app. It also gives the user more control over the app's functionality; for example, a user can choose to give a camera app access to the camera but not to the device location. The user can revoke the permissions at any time, by going to the app's Settings screen.

1. Start a new Android Studio project:
    - Name: MediaAccess
    - Package name: comp5216.sydney.edu.au.mediaaccess

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

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

<queries>
    <!-- Camera -->
    <intent>
        <action android:name="android.media.action.IMAGE_CAPTURE" />
    </intent>
    <!-- Gallery -->
    <intent>
        <action android:name="android.intent.action.GET_CONTENT" />
    </intent>
</queries>

</manifest>
```

2. Open AndroidManifest.xml, then add the following code outside the application block to declare that your app needs some permission:

3. If your **targetSdkVersion is 24 or higher**, we have to use FileProvider class to give access to the particular file or folder to make them accessible for other apps. We create our own class inheriting FileProvider in order to make sure our FileProvider doesn't conflict with FileProviders declared in imported dependencies. The detailed information can be found on https://commonsware.com/blog/2017/06/27/fileprovider-libraries.html

   Add a provider tag in AndroidManifest.xml under application tag. Specify a unique authority for the android:authorities attribute to avoid conflicts, since imported dependencies might specify ${applicationId}.provider and

```
<manifest>
        ...

        <application>
             ...
             <provider
                 android:name="androidx.core.content.FileProvider"
                 android:authorities="YOUR_APPLICATION_ID.fileProvider"
                 android:grantUriPermissions="true"
                 android:exported="false">
                 <meta-data
                     android:name="android.support.FILE_PROVIDER_PATHS"
                     android:resource="@xml/file_paths" />
             </provider>
             ...
        </application>

</manifest>
```

   other commonly used authorities.

4. Create a provider ***file_paths.xml*** file in the res/xml folder. The xml folder may need to be created if it doesn't exist. The content of the file is shown below. It describes that we would like to share access to External Storage directories with the relevant names.

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="root" path="." />
    <external-path name="my_images" path="/images/" />
    <external-path name="my_videos" path="/videos/" />
    <external-path name="my_audios" path="/audios/" />
</paths>
```

5. So far, we have finished setting up the necessary environment permission and for accessing the FileProvider.

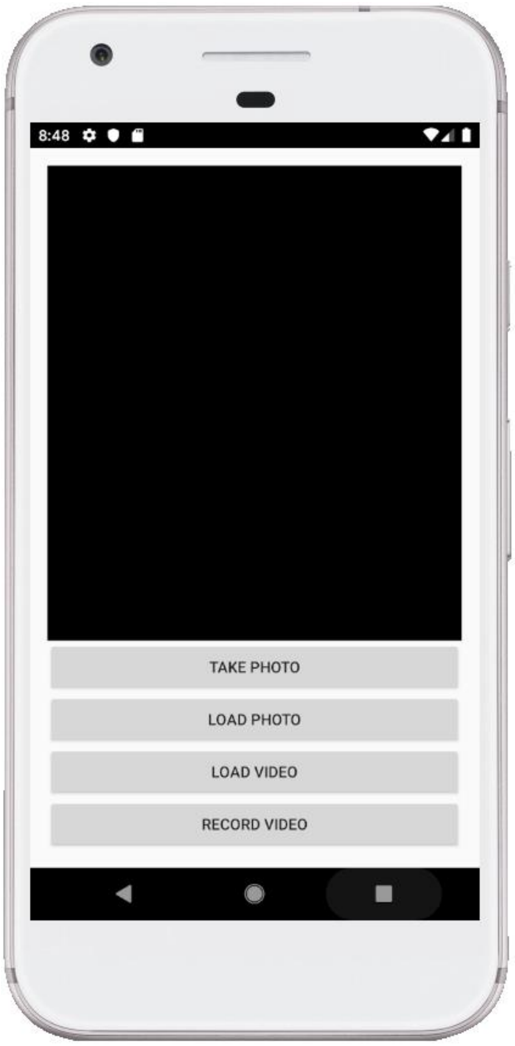
For further readings about Android runtime permissions and FileUriExposedException, please visit the following links:

https://developer.android.com/training/permissions/requesting

https://developer.android.com/reference/android/os/FileUriExposedException

Task 2: Use a basic layout

This layout has four buttons at the bottom and a VideoView and ImageView (overlapping each other) on the top.



1. The root layout is a RelativeLayout
2. Add a button for “**Record Video**” at the bottom using android:layout_alignParentBottom=“true”.
   Also assign an id to it: @+id/recordvideo.
3. Add a button for “**Load Video**” above the “**Record Video**” button using android:layout_above="@+id/recordvideo"
4. Do the same for “**Load Photo**” and “**Take Photo**”
5. Add a VideoView above the “**Take Photo**” button. Also add an ImageView above the “**Take Photo**” button.
6. If you are stuck, have a look at the sample layout file **activity_week05.xml** (in lab files).

Task 3: Take a photo using the phone's built-in Camera app

1. In a new Activity, use the above layout. Define a few global request codes. The

```
public final String APP_TAG = "MobileComputingTutorial";
public String photoFileName = "photo.jpg";
public String videoFileName = "video.mp4";
public String audioFileName = "audio.3gp";

//request codes
private static final int MY_PERMISSIONS_REQUEST_OPEN_CAMERA = 101;
private static final int MY_PERMISSIONS_REQUEST_READ_PHOTOS = 102;
private static final int MY_PERMISSIONS_REQUEST_RECORD_VIDEO = 103;
private static final int MY_PERMISSIONS_REQUEST_READ_VIDEOS = 104;
private static final int MY_PERMISSIONS_REQUEST_RECORD_AUDIO = 105;
```

values can be any integer, but each one must be different from others.

2. Add a helper method. Given a filename and type, this method returns the Uri for the filename, which will be generated when a photo is taken, or a video is recorded.

```
// Returns the Uri for a photo/media stored on disk given the fileName and type
public Uri getFileUri(String fileName, int type) {

    Uri fileUri = null;
    try {
        String typestr = "images"; //default to images type
        if (type == 1) {
            typestr = "videos";
        } else if (type != 0) {
            typestr = "audios";
        }

        // Get safe media storage directory depending on type
         File mediaStorageDir = new
       File(getExternalFilesDir(Environment.getExternalStorageDirectory().toStr
       ing()), APP_TAG);


        // Create the storage directory if it does not exist
        if (!mediaStorageDir.exists()) {
            mediaStorageDir.mkdirs();
        }

        // Create the file target for the media based on filename
        file = new File(mediaStorageDir, fileName);

        // Wrap File object into a content provider, required for API >= 24
        // See https://guides.codepath.com/android/Sharing-Content-with-
Intents#sharing-files-with-api-24-or-higher
        if (Build.VERSION.SDK_INT >= 24) {
            fileUri = FileProvider.getUriForFile(
                    this.getApplicationContext(),
                    "au.edu.sydney.comp5216.mediaaccess.fileProvider", file);
        } else {
            fileUri = Uri.fromFile(mediaStorageDir);
        }
    } catch (Exception ex) {
        Log.d("getFileUri", ex.getStackTrace().toString());
    }
    return fileUri;
}
```

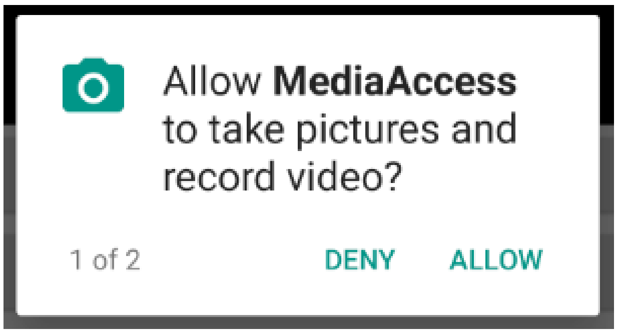
3. Copy the MarshmallowPermission.java to your class folder, then add an implementation in your MainActivity.java

```
MarshmallowPermission marshmallowPermission = new MarshmallowPermission(this);
```

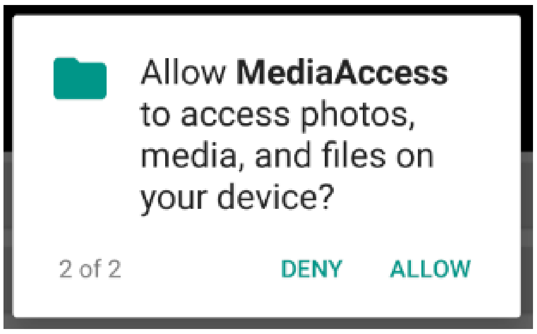
4. Add an onClick function for button "**Take Photo**", and link the following function with the layout xml file by adding attribute android:onClick="onTakePhotoClick" to the "**Take Photo**" button.

```
public void onTakePhotoClick(View v) {
    // Check permissions
    if (!marshmallowPermission.checkPermissionForCamera()) {
        marshmallowPermission.requestPermissionForCamera();
    }  else {
        // create Intent to take a picture and return control to the calling
application
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

        // set file name
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss",
                Locale.getDefault()).format(new Date());
        photoFileName = "IMG_" + timeStamp + ".jpg";

        // Create a photo file reference
        Uri file_uri = getFileUri(photoFileName,0);

        // Add extended data to the intent
        intent.putExtra(MediaStore.EXTRA_OUTPUT, file_uri);

        // If you call startActivityForResult() using an intent that no app can
handle, your app will crash.
        // So as long as the result is not null, it's safe to use the intent.
        if (intent.resolveActivity(getPackageManager()) != null) {
            // Start the image capture intent to take photo
            startActivityForResult(intent, MY_PERMISSIONS_REQUEST_OPEN_CAMERA);
        }
    }
}
```

Please note that you may request 2 different permissions (access to camera to take pictures and record video, and access to read/write to external storage) in this function. When you first run this app, you should get a dialog pop up that asks for your permission. Click "Allow" to grant permission.


Allow **MediaAccess** to take pictures and record video?

1 of 2 DENY ALLOW


Allow **MediaAccess** to access photos, media, and files on your device?

2 of 2 DENY ALLOW

5. Run it, preferably on a real phone. If running it on the emulator, the emulated camera will be used. But nothing happens when the photo is captured, because the onActivityResult() is not implemented yet.

Task 4: Select a photo from the gallery

1. Add an onClick function when the “**Load Photo**” button is clicked

```
public void onLoadPhotoClick(View view) {

        // Create intent for picking a photo from the gallery
        Intent intent = new Intent(Intent.ACTION_PICK,
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI);

        // Bring up gallery to select a photo
        startActivityForResult(intent, MY_PERMISSIONS_REQUEST_READ_PHOTOS);
 }
```

2. Add attribute android:onClick=“onLoadPhotoClick" to the “**Load Photo**” button

Task 5: Select a video from the gallery

1. Add a method when the “**Load Video**” button is clicked

```
public void onLoadVideoClick(View view) {
        // Create intent for picking a video from the gallery
        Intent intent = new Intent(Intent.ACTION_PICK,
                MediaStore.Video.Media.EXTERNAL_CONTENT_URI);

        // Bring up gallery to select a video
        startActivityForResult(intent, MY_PERMISSIONS_REQUEST_READ_VIDEOS);
}
```

2. Add attribute android:onClick=“onLoadVideoClick" to the “**Load Video**” button.

Task 6: Record a video

```
public void onRecordVideoClick(View v) {
    // Check permissions
    if (!marshmallowPermission.checkPermissionForCamera()
            || !marshmallowPermission.checkPermissionForExternalStorage()) {
        marshmallowPermission.requestPermissionForCamera();
    }  else {
        // create Intent to capture a video and return control to the calling
application
        Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

        // set file name
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss",
                Locale.getDefault()).format(new Date());
        videoFileName = "VIDEO_" + timeStamp + ".mp4";

        // Create a video file reference
        Uri file_uri = getFileUri(videoFileName,1);

        // add extended data to the intent
        intent.putExtra(MediaStore.EXTRA_OUTPUT, file_uri);

        // Start the video record intent to capture video
        startActivityForResult(intent, MY_PERMISSIONS_REQUEST_RECORD_VIDEO);
    }
}
```

1. Add a method to trigger when the “**Record Video**” button is clicked

2. Add attribute android:onClick=“onRecordVideoClick" to the “**Record Video**” button.

Task 7: Handle the media files

1. Add the onActivityResult() method to handle the results from the above

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    final VideoView mVideoView = (VideoView) findViewById(R.id.videoview);
    ImageView ivPreview = (ImageView) findViewById(R.id.photopreview);
    mVideoView.setVisibility(View.GONE);
    ivPreview.setVisibility(View.GONE);
    if (requestCode == MY_PERMISSIONS_REQUEST_OPEN_CAMERA) {
        if (resultCode == RESULT_OK) {
            // by this point we have the camera photo on disk
            Bitmap takenImage = BitmapFactory.decodeFile(file.getAbsolutePath());
            // Load the taken image into a preview
            ivPreview.setImageBitmap(takenImage);
            ivPreview.setVisibility(View.VISIBLE);
        } else { // Result was a failure
            Toast.makeText(this, "Picture wasn't taken AAA!",
                    Toast.LENGTH_SHORT).show();
        }
    } else if (requestCode == MY_PERMISSIONS_REQUEST_READ_PHOTOS) {
        if (resultCode == RESULT_OK) {
            Uri photoUri = data.getData();
            Bitmap selectedImage;
            try {
                selectedImage = MediaStore.Images.Media.getBitmap(
                        this.getContentResolver(), photoUri);
                ivPreview.setImageBitmap(selectedImage);
                ivPreview.setVisibility(View.VISIBLE);
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    } else if (requestCode == MY_PERMISSIONS_REQUEST_READ_VIDEOS) {
        if (resultCode == RESULT_OK) {
            Uri videoUri = data.getData();
            mVideoView.setVisibility(View.VISIBLE);
            mVideoView.setVideoURI(videoUri);
            mVideoView.requestFocus();
            mVideoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
                // Close the progress bar and play the video
                public void onPrepared(MediaPlayer mp) {
                    mVideoView.start();
                }
            });}
    } else if (requestCode == MY_PERMISSIONS_REQUEST_RECORD_VIDEO) {
//if you are running on emulator remove the if statement
        if (resultCode == RESULT_OK) {
            Uri takenVideoUri = getFileUri(videoFileName,1);
            mVideoView.setVisibility(View.VISIBLE);
            mVideoView.setVideoURI(takenVideoUri);
            mVideoView.requestFocus();
            mVideoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
                // Close the progress bar and play the video
                public void onPrepared(MediaPlayer mp) {
                    mVideoView.start();
                }
            });
```

actions. Read the inline comments.

2. For a photo, the VideoView is hidden, and the photo is placed in the ImageView. For a video, the ImageView is hidden, and the video is started in the VideoView.

3. Run it. The final code will be the same as that in **ActivityWeek05.java** (in lab files).

Task 8: Audio recording

This task is based on the following tutorial:
https://developer.android.com/guide/topics/media/mediarecorder

The Android multimedia framework includes support for capturing and encoding a variety of common audio formats, so that you can easily integrate audio into your applications. You can record audio using the **MediaRecorder** APIs if supported by the device hardware.

This document shows you how to write an application that captures audio from a device microphone, save the audio and play it back.

Note: The Android Emulator does not have the ability to record audio, but real devices are likely to provide these capabilities so be sure to test your code on a real device that can record.

Audio capture from the device is a bit more complicated than audio and video playback, but still fairly simple:

1. Create a new instance of **android.media.MediaRecorder**.
2. Set the audio source using **MediaRecorder.setAudioSource()**. You will probably want to use **MediaRecorder.AudioSource.MIC**.
3. Set output file format using **MediaRecorder.setOutputFormat()**.
4. Set output file name using **MediaRecorder.setOutputFile()**.
5. Set the audio encoder using **MediaRecorder.setAudioEncoder()**.
6. Call **MediaRecorder.prepare()** on the **MediaRecorder** instance.
7. To start audio capture, call **MediaRecorder.start()**.
8. To stop audio capture, call **MediaRecorder.stop()**.
9. When you are done with the **MediaRecorder** instance, call **MediaRecorder**.**release**() on it. Calling **MediaRecorder**.**release**() is always recommended to free the resources immediately.

Please refer to **AudioRecordTest.java** in lab files for sample code.

Task 9: Implement your own camera interface

If you want to embed a camera view in your own app instead of using the built-in Camera support, read the following explanation on Camera API, which discusses a quick, simple approach to image and video capture, and also outlines an advanced approach for creating custom camera experiences for your users:
https://developer.android.com/guide/topics/media/camera

Some contents are adapted from:

https://developer.android.com/training/camera

https://github.com/codepath/android_guides/wiki/Accessing-the-Camera-and-Stored-Media