**Android Multi-media API :-**

Multimedia API is represented by android.media and android.widget packages and android Multimedia API contains all needed classes and interfaces that are required to deal with multi media content like audio, video files.

**Q. How to play and control an audio file in Android?**

**Ans.**  Android provides many ways to control playback of an audio file. One of the way is through a class called MediaPlayer.

**Media player:** This class has been defined in android.media pkg. An object of this class provides facilities to access built in Media player Services for playing & controlling audio files.

An object of MediaPlayer class is created by

Static MediaPlayer create(Context context, int id); method of this class.

Eg-

MediaPlayer mb= MediaPlayer.create(this,R.raw.songName)

We have to use this approach while we are playing an audio file which already has stored in application/ project. (eg. Game).

**Location of song file:-** projectName/res/raw/song.mp3

**OR**

Through “new” keyword

We can also create an object of this class by using of “new” keyword of default Constructor calling.

MediaPlayer mp= new MediaPlayer();

**Note:- This approach is used when we have the extract song from external storage.**

**METHODS OF MEDIAPLAYER CLASS**

1. **public void setDataSource(String Path) :-**  set the data source (file path or http url) to use.
2. **public void prepare() :-**  prepares the player for playback synchronously.
3. **public void start() :-**  it starts or resumes the playback.
4. **public void stop():-** it stops the play back.
5. **public void pause() :-** it pauses the play back.
6. **public boolean isPlaying() :-** checks if Media Player is playing.
7. **public void seekTo(int millis) :-**  seeks to specified time in milliseconds.
8. **public void setLooping(boolean looping) :-**  sets the player for looping or non-looping.
9. **public void selectTrack(int index) :-** it selects a track for specified index.
10. **public int getCurrentPosition() :-** returns the current playback position.
11. **public int getDuration():-** returns duration of the file.
12. **public void setVolume(float leftVolume, float rightVolume):-**  sets the volume of the player.

**Note:-** we can also set a view containing controls(play/pause, prev/next etc) for MediaPlayer by MediaController.

**MediaController Class:-** This class is define in android.widget pkg. An object of this class provides controls for media (MediaPlayer/VideoView) which typically contains the buttons like "Play/Pause", "Rewind", "Fast Forward" and a progress slider.

The way to use this class is to instantiate it programmatically. The MediaController will create a default set of controls and put them in a window floating above the view specified with **setAnchorView()**. The window will disappear if left idle for three seconds and reappear when the user touches the anchor view.

Functions like show() and hide() have no effect when MediaController is created in an xml layout. MediaController will hide and show the buttons according to these rules:

* The "previous" and "next" buttons are hidden until setPrevNextListeners() has been called
* The "previous" and "next" buttons are visible but disabled if setPrevNextListeners() was called with null listeners
* The "rewind" and "fastforward" buttons are shown unless requested otherwise by using the MediaController(Context, boolean) constructor with the boolean set to false

The way to use this class is to instantiate it programmatically.

**MediaController m= new MediaController(Context context);**

**Important Methods –**

|  |  |
| --- | --- |
| void | [setAnchorView](http://developer.android.com/reference/android/widget/MediaController.html#setAnchorView(android.view.View))([View](http://developer.android.com/reference/android/view/View.html) view)  Set the view that acts as the anchor for the control view. |
| void | [setEnabled](http://developer.android.com/reference/android/widget/MediaController.html#setEnabled(boolean))(boolean enabled)  Set the enabled state of this view. |
| void | [setMediaPlayer](http://developer.android.com/reference/android/widget/MediaController.html#setMediaPlayer(android.widget.MediaController.MediaPlayerControl))([MediaController.MediaPlayerControl](http://developer.android.com/reference/android/widget/MediaController.MediaPlayerControl.html) player) |
| void | [setPrevNextListeners](http://developer.android.com/reference/android/widget/MediaController.html#setPrevNextListeners(android.view.View.OnClickListener, android.view.View.OnClickListener))([View.OnClickListener](http://developer.android.com/reference/android/view/View.OnClickListener.html) next, [View.OnClickListener](http://developer.android.com/reference/android/view/View.OnClickListener.html) prev) |
| void | [show](http://developer.android.com/reference/android/widget/MediaController.html#show(int))(int timeout)  Show the controller on screen. |
| void | [show](http://developer.android.com/reference/android/widget/MediaController.html#show())()  Show the controller on screen. |
| boolean | [isShowing](http://developer.android.com/reference/android/widget/MediaController.html#isShowing())() |

**Q. How to play and control video file in Android?**

**Ans.** Android provides facilities to play and control any video file with the help of “VideoView” and “MediaController” classes.

Both classes are define in android.widget packages.

**VideoView:-** An Object of this class provides facility to play and control video player with the help of Mediacontroller.

In order to use video view, we have to define <VideoView> tag in .xml file.

**METHODS OF VIDEO VIEW:-**

1. **public void setMediaController(MediaController Controller):-**  sets the MediaController on VideoView.
2. **public void setVideoURI(Uri uri):-**  set the URI of the video file.
3. **public void start():-**  start the video view.
4. **public void stopPlayBack :-**  stops the playback.
5. **public void pause():-**  pauses the playback.
6. **public void suspend():**  suspend the playback.
7. **public void resume():-** resumes the playback.
8. **public void seek(int mills):-** seek to specified time in milliseconds.

**Q. How to record audio/video in android?**

**Ans.** Every android device has a built in Micro-Phone through which we can record audio and store it, or play it over device/phone. There are many ways to do it but the most common way is through **“MediaRecorder”** class.

**MediaRecorder:-**  This class has been defined in android.media pkg. An object of this class provides facilities to record audio/video and create media file to play later etc.

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 capture audio, but actual devices are likely to provide these capabilities.

**How to instantiate MediaRecorder?**

**MediaRecorder mr= new MediaRecorder();**

**NOTE:- After recording Media, we can create a sound files that can be played later.**

After creating an object of MediaRecorder class, we have to set the Audio receiving source, output format , encoding format and output file’s location.

**SYNTAX:-**

**mr.setAudioSource(MediaRecorder.AudioSource.MIC);**

**Set o/p format:-**

**mr.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);**

**Set audio encoder:-**

**mr.setAudioEncoder(MediaRecorder.OutputFormat.AMR\_NB);**

**Save the o/p Files:-**

**mr.setOutputFile(outputFile.getAbsolutePath());**

After specifying AudioSource, Format and its o/p files, we have to call two basic following methods on MediaRecorder class:-

1. mr.prepare();
2. mr.start();

**NOTE:- A part from these Methods , there are other methods in MediaRecorder Class that allows us to more control over Audio/Video Recording.**

**COMMONLY USED METHODS:-**

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 configure 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 record instance is not needed.

**NOTE:- we have to take following permission from Android OS by defining in Android.Manifest.xml :-**

**<use permission name= “android.permission.WRITE\_EXTERNAL\_STORAGE >;**

**<use permission name= “android.permission.RECORD\_AUDIO”>;**

**Android camera API :-** Almost android devices have at least one camera. Some devices have front camera and back camera as well.

Using the cameraandroid devices can be done by following two ways by which we can use camera in our application :-

1. **Using existing android camera application in our application.**
2. **We can build our own camera application by using camera API(android.hardware.camera2 pkg added in API 21).**

**Using Existing Camera Application :-** In this case, we nothing to do much except sending/ broadcasting an implicit intent to inbuilt android camera application.

We must use following Action in implicit intent for using inbuilt camera app-

android.provider.MediaStore.ACTION\_IMAGE\_CAPTURE.

OR.

ACTION\_VIDEO\_CAPTURE to launch an existing camera application install on our device.

**Intent i= new Intent(android.provider.MediaStore.ACTION\_IMAGE\_CAPTURE);**

**Working with inbuilt CAMERA application we need to set permission and features in manifest.xml file. So that we have to add following features and permission in android manifest file.**

<use\_permission name=”android.permission.WRITE\_EXTERNAL\_STORAGE”/>

< use\_permission name=” android.permission.CAMERA”/>

**And We should check whether CAMERA is available or not on device as following-**

## 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() {

if (getPackageManager().hasSystemFeature(

PackageManager.FEATURE\_CAMERA\_FRONT)){

return true;

} else {

return false;

}

}

## 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. A specific location for the media file may be specified using the putExtra() method of the intent, referencing the MediaStore.EXTRA\_OUTPUT key constant to pass through the target URI value. The following code, for example, specifies that the video should be stored on the SD card in a file named myvideo.mp4:

File mediaFile =

new File(Environment.getExternalStorageDirectory().getAbsolutePath()

+ "/myvideo.mp4");

Intent intent = new Intent(MediaStore.ACTION\_VIDEO\_CAPTURE);

Uri videoUri = Uri.fromFile(mediaFile);

intent.putExtra(MediaStore.EXTRA\_OUTPUT, videoUri);

startActivityForResult(intent, VIDEO\_CAPTURE);

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:

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == VIDEO\_CAPTURE) {

if (resultCode == RESULT\_OK) {

Toast.makeText(this, "Video saved to:\n" +

data.getData(), 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();

}

}

}