4.4 AUDIO IMPLEMENTATIONS IN ANDROID

To learn how to play audio, we create a new project named "MediaT- ester" and keep other configuration default. Then we copy one song from local directory to "MediaTester\app\src\main\res\raw" directory. Notice that we need to ensure that the file format can be recognized by Android. Fortunately, Android supports most all kinds of audio file formats. However, if Android does not support the file format of your audio, try to transform it to a common format.

First, create two buttons to show the "start" and "pause" functions. As introduced in the previous chapter, we create two buttons in the activity_main.xml, as shown in Fig. 4.22. Meanwhile, we need to add two strings in the strings.xml file as:

< *stringname* = "start_button" > *Start* < */string* >

< *stringname* = "pause_button" > *P ause* < */string* >

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/start_button"
    android:id="@+id/button_start"
    android:layout_below="@+id/textView"
    android:layout_alignParentStart="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pause_button"
    android:id="@+id/button_pause"
    android:layout_below="@+id/button_start"
    android:layout_alignParentStart="true" />
```

Figure 4.22 Creating two buttons in activity_main.xml.

Then jump into the MainActivity.java file and add a new Medi- aPlayer object into the MainActivity class as:

private MediaPlayer mp

Then we modify the MainActivity to implement OnClickListener, as introduced in

previous chapter, and create onClick(View v) method to implement the functions of these two buttons. Then set OnClickLis- tener to these two buttons, and now the MainActivity.java is shown as Fig. 4.23.

```
public class MainActivity extends Activity implements View.OnClickListener {

    private MediaPlayer mp;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        findViewById(R.id.button_1).setOnClickListener(this);
        findViewById(R.id.button_2).setOnClickListener(this);
    }

    public void onClick(View v)
    {...}
```

Figure 4.23 MediaPlayer object and the OnClickListener.

Then in the *onClick()* method, we implement functions to these two buttons, which are start a song, pause, and resume it. Before starting a song, we need to create a resource to the *MediaPlayer* object. Then we need to tell the computer which audio we want to play. We can use an integer ID of audio resource to identify the audio resource. In our example, we use resId = R.raw.test, then we call the *start()* method to play music. Before use pause a song, we need to judge whether it is playing. If it is playing, we call *pause()* method to pause it; if not, we call *start()* method to resume it. The code is shown in Fig. 4.24.

The running result is shown in Fig. 4.25. When we click the “START” button, Android plays the song that we previously put in the raw file. When we click the “PAUSE” button, Android will pause the song if it is playing or resume it if it is paused.

4.5 EXECUTING VIDEO IN ANDROID

The MediaPlayer class works with video the same way it does with audio. However, we need to create a surface to play video, and the surface is VideoView class. The VideoView class can load images from various sources and takes charge of computing its

measurement from the video. Furthermore, it provides various display options, such as scaling.

HINT: VideoView does not retain its full state when going into the background. It does not restore the current play state, position, selected tracks, or any subtitle tracks.

We will add something about video into the MediaTester project.

```
public void onClick(View v) {
    int resId;
    switch (v.getId()) {
        case R.id.button_start:
            resId = R.raw.test;
            if (mp != null) {
                mp.release();
            }
            mp = MediaPlayer.create(this, resId);
            mp.start();
            break;
        case R.id.button_pause:
            if (mp.isPlaying()) {
                mp.pause();
            } else {
                mp.start();
            }
            break;
    }
```

Figure 4.24 Implementing the functions of two buttons in onClick() method.

Figure 4.25 Running result of the MediaTester.

First, we create a new VideoView below the pause button in

activity_main.xml as follows:

<VideoView

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:id="@+id/video"

android:layout_below="@+id/button_pause"

android:layout_gravity="center"/>

Then, jumping into Java file, we create a View object named video and connect it to the VideoView as follows:

```
VideoView video = (VideoView) findViewById(R.id.video);
```

Then we need to set a path to identify the location of the video. However, the *Android Virtual Device* (AVD) cannot recognize the local path in our computer. Android offers several options to store persistent application data as follows:

Shared Preferences The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types.

Internal Storage We can save files directly on the device's internal storage. Files saved to the internal storage are private in default.

External Storage Android devices support a shared external stor- age to save files. The external storage can be a removable storage media, such as an SD card, or internal storage.

SQLite Databases We can use SQLite in Android to create databases that will be accessible by name to any class in the app.

Network Connection We can use the network to store and retrieve data in our own services.

Before we play a video using VideoView, we need to set a path to locate the video, and

this path must be inside the AVD itself.

First of all, run an AVD, and jump into Dalvik Debug Monitor Service (DDMS) after the AVD runs. In Android Studio, select Tools, then Android, then click Android Device Monitor (Tools → Android → Android Device Monitor), as shown in Fig. 4.26. The Android Device Monitor will be similar to Fig. 4.27.

Then select the AVD we just run, and then in the "File Explorer" tab, we can see many folders and files listed. Find the "data" folder and click "Push a file onto the device" on the right top of the interface, as shown in Fig. 4.28. Then select and push a local video file onto the device.

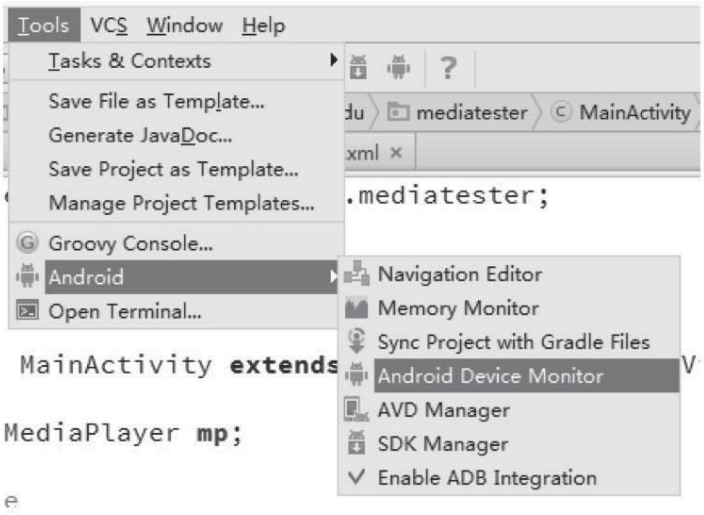

Figure 4.26 Android device monitor.

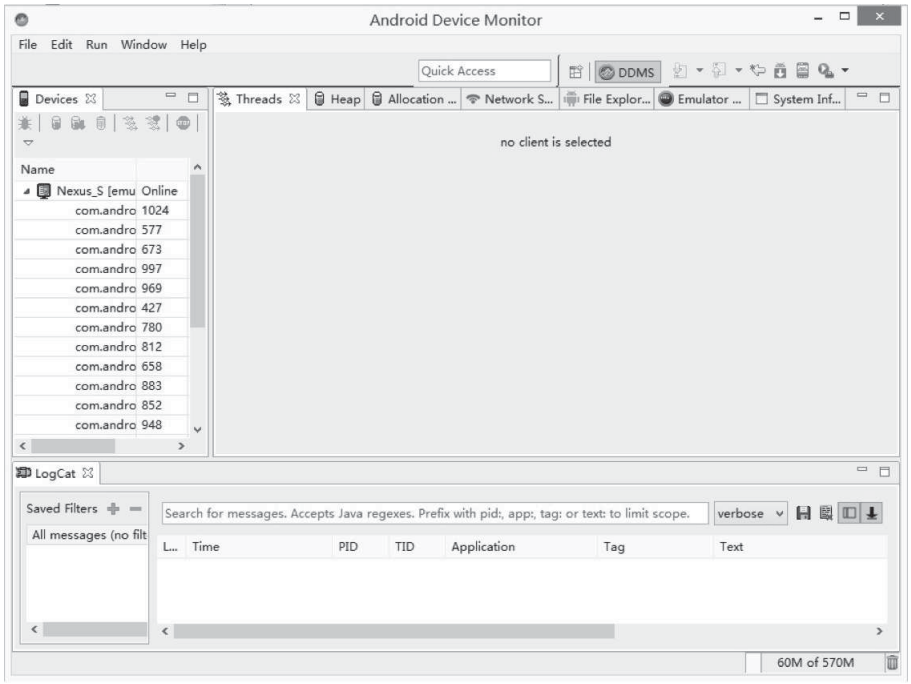

Figure 4.27 Android device Monitor.

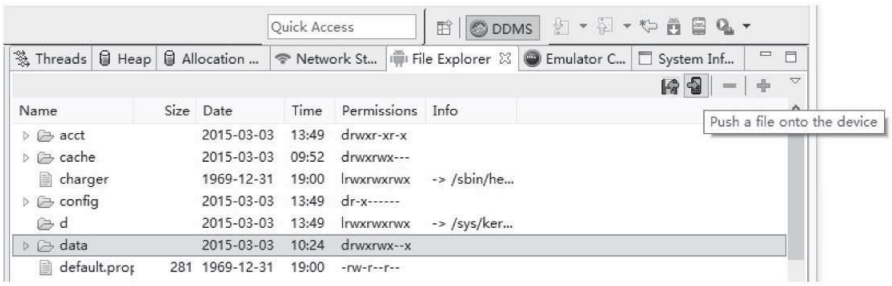

Figure 4.28 Push a file onto the device.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //audio part
    findViewById(R.id.button_start).setOnClickListener(this);
    findViewById(R.id.button_pause).setOnClickListener(this);

    //video part
    //Before we play a video, we need to push the video resource
    //into AVD
    VideoView video = (VideoView) findViewById(R.id.video);
    video.setVideoPath("/data/samplevideo.3gp");
    video.start();
}
```

Figure 4.29 Setting video path and start a video.

The DDMS is used to operate the AVD, not the Android app. If we have pushed some video into a device before, we do not need to push it again in Android Project. Then add

two methods to the onCreate() method to set the Video path and play it as follows:

video.setVideoPath("/data/samplevideo.3gp"); video.start();

Then the current onCreate() method can refer to Fig. 4.29.

In the end, the running result is shown in Fig. 4.30.

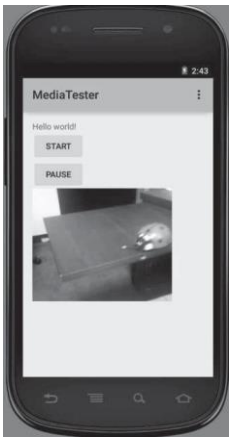

Figure 4.30 The running result of the MediaTester.