CHAPTER 7

Playing Media

Pretty much every phone claiming to be a "smartphone" has the ability to at least play back music, if not video. Even many more ordinary phones are full-fledged MP3 players, in addition to offering ringtones and whatnot.

Not surprisingly, Android has multimedia support for you, as a developer, to build your own games, media players, and so on.

This chapter is focused on audio and video playback; other chapters will tackle media input, including the camera and audio recording.

Get Your Media On

In Android, you have five different places you can pull media clips from – one of these will hopefully fit your needs:

1. You can package media clips as raw resources (`res/raw` in your project), so they are bundled with your application. The benefit is that you're guaranteed the clips will be there; the downside is that they cannot be replaced without upgrading the application.
2. You can package media clips as assets (`assets/` in your project) and reference them via `file:///android_asset/` URLs in a `Uri`. The benefit over raw resources is that this location works with APIs that expect `Uri` parameters instead of resource IDs. The downside –

assets are only replaceable when the application is upgraded – remains.

3. You can store media in an application-local directory, such as content you download off the Internet. Your media may or may not be there, and your storage space isn't infinite, but you can replace the media as needed.
4. You can store media – or reference media that the user has stored herself – that is on an SD card. There is likely more storage space on the card than there is on the device, and you can replace the media as needed, but other applications have access to the SD card as well.
5. You can, in some cases, stream media off the Internet, bypassing any local storage, as with the StreamFurious application

Internet streaming is tricky, particularly for video, and is well beyond the scope of this book. For the T-Mobile G1, the recommended approach for anything of significant size is to put it on the SD card, as there is very little on-board flash memory for file storage.

Making Noise

The crux of playing back audio comes in the form of the `MediaPlayer` class. With it, you can feed it an audio clip, start/stop/pause playback, and get notified on key events, such as when the clip is ready to be played or is done playing.

You have three ways to set up a `MediaPlayer` and tell it what audio clip to play:

1. If the clip is a raw resource, use `MediaPlayer.create()` and provide the resource ID of the clip
2. If you have a `Uri` to the clip, use the `Uri`-flavored version of `MediaPlayer.create()`
3. If you have a string path to the clip, just create a `MediaPlayer` using the default constructor, then call `setDataSource()` with the path to the clip

Next, you need to call `prepare()` or `prepareAsync()`. Both will set up the clip to be ready to play, such as fetching the first few seconds off the file or stream. The `prepare()` method is synchronous; as soon as it returns, the clip is ready to play. The `prepareAsync()` method is asynchronous – more on how to use this version later.

Once the clip is prepared, `start()` begins playback, `pause()` pauses playback (with `start()` picking up playback where `pause()` paused), and `stop()` ends playback. One caveat: you cannot simply call `start()` again on the `MediaPlayer` once you have called `stop()` – we'll cover a workaround a bit later in this section.

To see this in action, take a look at the `Media/Audio` sample project. The layout is pretty trivial, with three buttons and labels for play, pause, and stop:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="4px"
  >
    <ImageButton android:id="@+id/play"
      android:src="@drawable/play"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:paddingRight="4px"
      android:enabled="false"
    />
    <TextView
      android:text="Play"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_vertical"
      android:layout_gravity="center_vertical"
      android:textAppearance="?android:attr/textAppearanceLarge"
    />
  </LinearLayout>
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="4px"
  >
    <ImageButton android:id="@+id/pause"
      android:src="@drawable/pause"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:paddingRight="4px"
    />
    <TextView
      android:text="Pause"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_vertical"
      android:layout_gravity="center_vertical"
      android:textAppearance="?android:attr/textAppearanceLarge"
    />
  </LinearLayout>
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="4px"
  >
    <ImageButton android:id="@+id/stop"
      android:src="@drawable/stop"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:paddingRight="4px"
    />
    <TextView
      android:text="Stop"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_vertical"
      android:layout_gravity="center_vertical"
      android:textAppearance="?android:attr/textAppearanceLarge"
    />
  </LinearLayout>
</LinearLayout>
```

The Java, of course, is where things get interesting:

```
public class AudioDemo extends Activity
  implements MediaPlayer.OnCompletionListener {

  private ImageButton play;
  private ImageButton pause;
  private ImageButton stop;
  private MediaPlayer mp;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    play=(ImageButton)findViewById(R.id.play);
    pause=(ImageButton)findViewById(R.id.pause);
    stop=(ImageButton)findViewById(R.id.stop);

    play.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        play();
      }
    });

    pause.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        pause();
      }
    });

    stop.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        stop();
      }
    });

    setup();
  }

  @Override
  public void onDestroy() {
    super.onDestroy();

    if (stop.isEnabled()) {
      stop();
    }
  }

  public void onCompletion(MediaPlayer mp) {
    stop();
  }

  private void play() {
    mp.start();

    play.setEnabled(false);
    pause.setEnabled(true);
    stop.setEnabled(true);
  }

  private void stop() {
    mp.stop();
    mp.release();
    setup();
  }
  private void pause() {
    mp.pause();

    play.setEnabled(true);
    pause.setEnabled(false);
    stop.setEnabled(true);
  }

  private void loadClip() {
    try {
      mp=MediaPlayer.create(this, R.raw.clip);
      mp.setOnCompletionListener(this);
    }
    catch (Throwable t) {
      goBlooey(t);
    }
  }

  private void setup() {
    loadClip();
    play.setEnabled(true);
    pause.setEnabled(false);
    stop.setEnabled(false);
  }

  private void goBlooey(Throwable t) {
    AlertDialog.Builder builder=new AlertDialog.Builder(this);

    builder
      .setTitle("Exception!")
      .setMessage(t.toString())
      .setPositiveButton("OK", null)
      .show();
  }
}
```

In onCreate(), we wire up the three buttons to appropriate callbacks, then call setup(). In setup(), we create our MediaPlayer, set to play a clip we package in the project as a raw resource. We also configure the activity itself as the completion listener, so we find out when the clip is over. Note that, since we use the static create() method on MediaPlayer, we have already implicitly called prepare(), so we do not need to call that separately ourselves.

The buttons simply work the MediaPlayer and toggle each others' states, via appropriately-named callbacks. So, play() starts MediaPlayer playback, pause() pauses playback, and stop() stops playback and resets our

MediaPlayer to play again. The stop() callback is also used for when the audio clip completes of its own accord.

To reset the MediaPlayer, the stop() callback calls release() on the existing MediaPlayer (to release its resources), then calls setup() again, discarding the used MediaPlayer and starting a fresh one.

The UI is nothing special, but we are more interested in the audio in this sample, anyway:

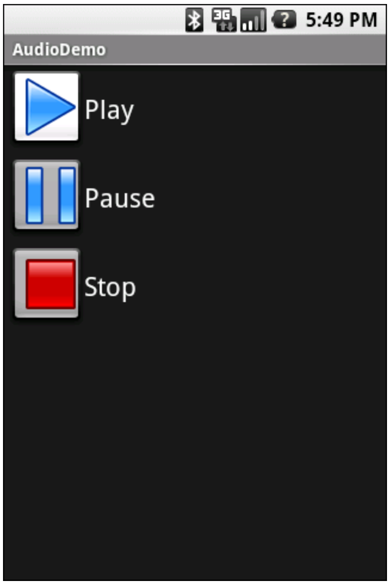

Figure 30. The AudioDemo sample application

Moving Pictures

Video clips get their own widget, the VideoView. Put it in a layout, feed it an MP4 video clip, and you get playback!

For example, take a look at this layout, from the Media/Video sample project:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <VideoView
    android:id="@+id/video"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
    />
</LinearLayout>
```

The layout is simply a full-screen video player. Whether it will use the full screen will be dependent on the video clip, its aspect ratio, and whether you have the device (or emulator) in portrait or landscape mode.

Wiring up the Java is almost as simple:

```
public class VideoDemo extends Activity {
  private VideoView video;
  private MediaController ctlr;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    getWindow().setFormat(PixelFormat.TRANSLUCENT);
    setContentView(R.layout.main);

    File clip=new File("/sdcard/test.mp4");

    if (clip.exists()) {
      video=(VideoView)findViewById(R.id.video);
      video.setVideoPath(clip.getAbsolutePath());

      ctlr=new MediaController(this);
      ctlr.setMediaPlayer(video);
      video.setMediaController(ctlr);
      video.requestFocus();
    }
  }
}
```

The biggest trick with `VideoView` is getting a video clip onto the device. While `VideoView` does support some streaming video, the requirements on the MP4 file are fairly stringent. If you want to be able to play a wider array of video clips, you need to have them on the device, preferably on an SD card.

The crude `VideoDemo` class assumes there is an MP4 file in `/sdcard/test.mp4` on your emulator. To make this a reality:

1. Find a clip, such as Aaron Rosenberg's *Documentaries and You* from Duke University's Center for the Study of the Public Domain's Moving Image Contest, which was used in the creation of this book
2. Use `mksdcard` (in the Android SDK's tools directory) to create a suitably-sized SD card image (e.g., `mksdcard 128M sd.img`)
3. Use the `-sdcard` switch when launching the emulator, providing the path to your SD card image, so the SD card is "mounted" when the emulator starts
4. Use the `adb push` command (or DDMS or the equivalent in your IDE) to copy the MP4 file into `/sdcard/test.mp4`

Once there, the Java code shown above will give you a working video player:

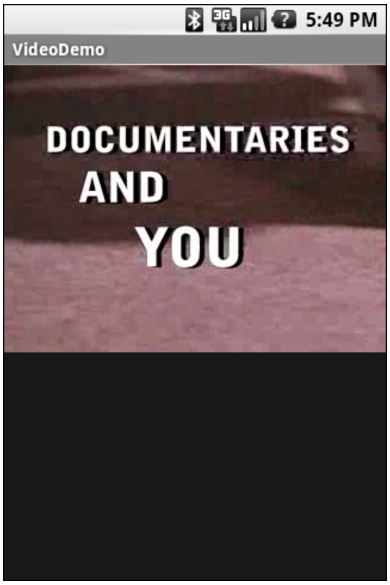

Figure 31. The VideoDemo sample application, showing a Creative Commons-licensed video clip

Tapping on the video will pop up the playback controls:

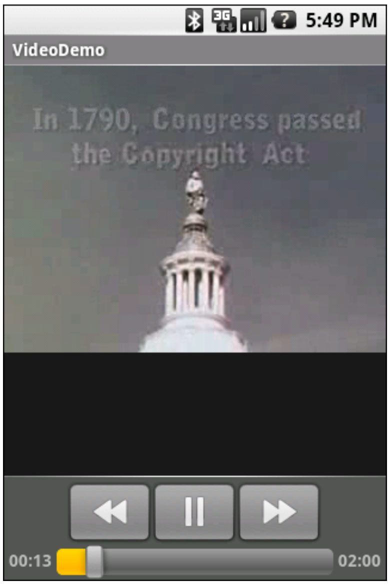

Figure 32. The VideoDemo sample application, with the media controls displayed

The video will scale based on space, as shown in this rotated view of the emulator (`<Ctrl>-<F12>`):

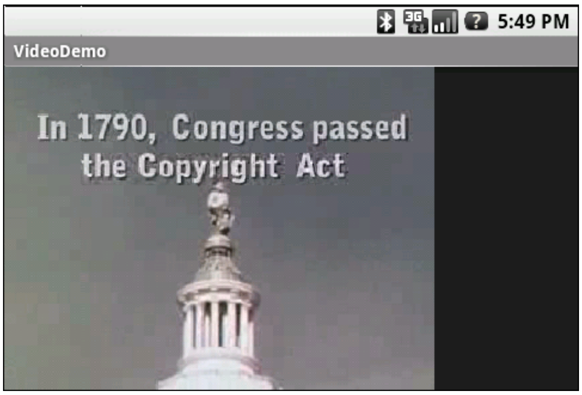

Figure 33. The VideoDemo sample application, in landscape mode, with the video clip scaled to fit

Note that playback may be rather jerky in the emulator, depending on the power of the PC that is hosting the emulator. For example, on a Pentium-M

1.6GHz PC, playback in the emulator is extremely jerky, while playback on the T-Mobile G1 is very smooth.