62. Implementing Video Playback on Android using the VideoView and MediaController Classes

One of the primary uses for smartphones and tablets is to enable the user to access and consume content. One key form of content widely used, especially in the case of tablet devices, is video.

The Android SDK includes two classes that make the implementation of video playback on Android devices extremely easy to implement when developing applications. This chapter will provide an overview of these two classes, VideoView and MediaController, before working through the creation of a simple video playback application.

62.1 Introducing the Android VideoView Class

By far the simplest way to display video within an Android application is to use the VideoView class. This is a visual component which, when added to the layout of an activity, provides a surface onto which a video may be played. Android currently supports the following video formats:

- H.263
- H.264 AVC
- H.265 HEVC
- MPEG-4 SP
- VP8
- VP9

The VideoView class has a wide range of methods that may be called in order to manage the playback of video. Some of the more commonly used methods are as follows:

- **setVideoPath(String path)** – Specifies the path (as a string) of the video media to be played. This can be either the URL of a remote video file or a video file local to the device.

- **setVideoUri(Uri uri)** – Performs the same task as the setVideoPath() method but takes a Uri object as an argument instead of a string.
- **start()** – Starts video playback.
- **stopPlayback()** – Stops the video playback.
- **pause()** – Pauses video playback.
- **isPlaying()** – Returns a Boolean value indicating whether a video is currently playing.
- **setOnPreparedListener(MediaPlayer.OnPreparedListener)** – Allows a callback method to be called when the video is ready to play.
- **setOnErrorListener(MediaPlayer.OnErrorListener)** - Allows a callback method to be called when an error occurs during the video playback.
- **setOnCompletionListener(MediaPlayer.OnCompletionListener)** - Allows a callback method to be called when the end of the video is reached.
- **getDuration()** – Returns the duration of the video. Will typically return -1 unless called from within the OnPreparedListener() callback method.
- **getCurrentPosition()** – Returns an integer value indicating the current position of playback.
- **setMediaController(MediaController)** – Designates a MediaController instance allowing playback controls to be displayed to the user.

62.2 Introducing the Android MediaController Class

If a video is simply played using the VideoView class, the user will not be given any control over the playback, which will run until the end of the video is reached. This issue can be addressed by attaching an instance of the MediaController class to the VideoView instance. The MediaController will then provide a set of controls allowing the user to manage the playback (such as pausing and seeking backwards/forwards in the video time-line).

The position of the controls is designated by anchoring the controller instance to a specific view in the user interface layout. Once attached and

anchored, the controls will appear briefly when playback starts and may subsequently be restored at any point by the user tapping on the view to which the instance is anchored.

Some of the key methods of this class are as follows:

- **setAnchorView(View view)** – Designates the view to which the controller is to be anchored. This controls the location of the controls on the screen.
- **show()** – Displays the controls.
- **show(int timeout)** – Controls are displayed for the designated duration (in milliseconds).
- **hide()** – Hides the controller from the user.
- **isShowing()** – Returns a Boolean value indicating whether the controls are currently visible to the user.

62.3 Creating the Video Playback Example

The remainder of this chapter is dedicated to working through an example application intended to use the VideoView and MediaController classes to play a web based MPEG-4 video file.

Create a new project in Android Studio, entering *VideoPlayer* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of an Empty Activity named *VideoPlayerActivity* with a corresponding layout named *activity_video_player*.

62.4 Designing the VideoPlayer Layout

The user interface for the main activity will consist solely of an instance of the VideoView class. Use the Project tool window to locate the *app -> res -> layout -> activity_video_player.xml* file, double-click on it, switch the Layout Editor tool to Design mode and delete the default TextView widget.

From the Images category of the Palette panel, drag and drop a VideoView

instance onto the layout so that it fills the available canvas area as shown in Figure 62-1. Using the Attributes panel, change the layout_width and layout_height attributes to *match_constraint* and *wrap_content* respectively. Also, remove the constraint connecting the bottom of the VideoView to the bottom of the parent ConstraintLayout. Finally, change the ID of the component to *videoView1*.

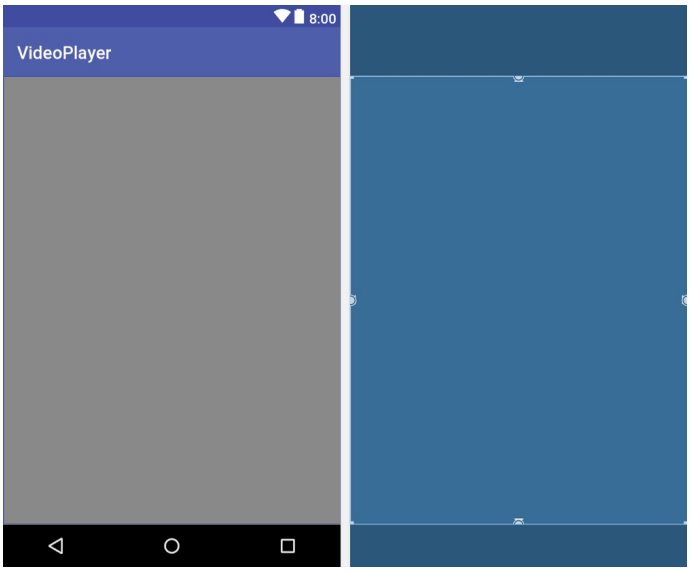

Figure 62-1

On completion of the layout design, the XML resources for the layout should read as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://sch
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.ebookfrenzy.videoplayer.VideoPlayerActivity"
    tools:layout_editor_absoluteX="0dp"
    tools:layout_editor_absoluteY="81dp">

    <VideoView
        android:id="@+id/videoView1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

62.5 Configuring the VideoView

The next step is to configure the VideoView with the path of the video to be played and then start the playback. This will be performed when the main activity has initialized, so load the *VideoPlayerActivity.java* file into the editor and modify the it as outlined in the following listing:

```
package com.ebookfrenzy.videoplayer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;

public class VideoPlayerActivity extends AppCompatActivity {

    private VideoView videoView;

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

    private void configureVideoView() {

        videoView =
                findViewById(R.id.videoView1);

        videoView.setVideoPath(
                "http://www.ebookfrenzy.com/android_book/movie.mp4");

        videoView.start();
    }
}
```

All that this code does is obtain a reference to the VideoView instance in the layout, set the video path on it to point to an MPEG-4 file hosted on a web site and then start the video playing.

62.6 Adding Internet Permission

An attempt to run the application at this point would result in the application failing to launch with an error dialog appearing on the Android device that reads “Unable to Play Video. Sorry, this video cannot be played”. This is not because of an error in the code or an incorrect video file format. The issue would be that the application is attempting to access a file over the internet, but has failed to request appropriate permissions to do so. To resolve this, edit the *AndroidManifest.xml* file for the project and add a line to request internet access:

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
.
.
.
</manifest>
```

Test the application by running it on a physical Android device. After the application launches there may be a short delay while video content is buffered before the playback begins (Figure 62-2).

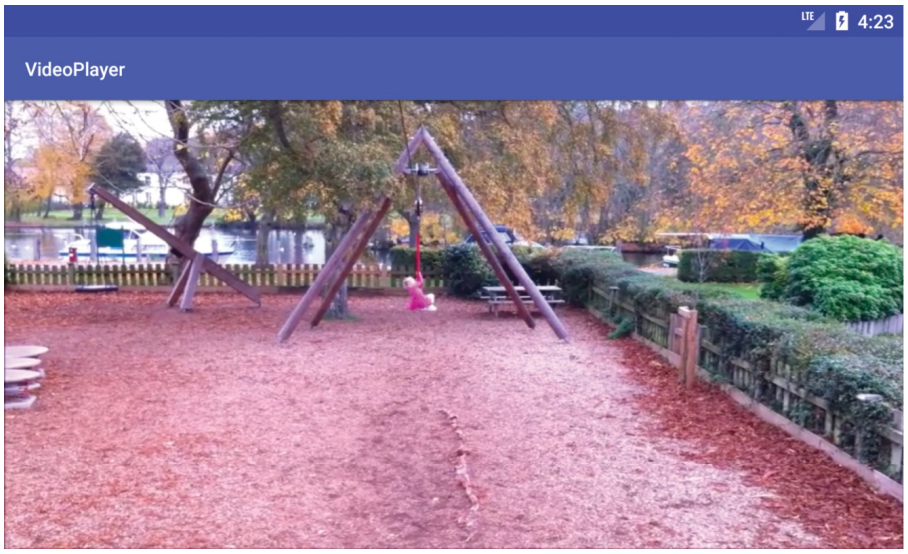

Figure 62-2

This provides an indication of how easy it can be to integrate video playback into an Android application. Everything so far in this example has been achieved using a VideoView instance and three lines of code.

62.7 Adding the MediaController to the Video View

As the VideoPlayer application currently stands, there is no way for the user to control playback. As previously outlined, this can be achieved using the MediaController class. To add a controller to the VideoView, modify the *configureVideoView()* method once again:

```
package com.ebookfrenzy.videoplayer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;
import android.widget.MediaController;

public class VideoPlayerActivity extends AppCompatActivity {

        private VideoView videoView;
        private MediaController mediaController;
.
.
    private void configureVideoView() {

         final VideoView videoView =
                 findViewById(R.id.videoView1);
```

```
        videoView.setVideoPath(
                "http://www.ebookfrenzy.com/android_book/movie.mp4");

        mediaController = new MediaController(this);
        mediaController.setAnchorView(videoView);
        videoView.setMediaController(mediaController);

        videoView.start();

    }
}
```

When the application is launched with these changes implemented, tapping the VideoView canvas will cause the media controls will appear over the video playback. These controls should include a seekbar together with fast forward, rewind and play/pause buttons. After the controls recede from view, they can be restored at any time by tapping on the VideoView canvas once again. With just three more lines of code, our video player application now has media controls as shown in Figure 62-3:

Figure 62-3

62.8 Setting up the onPreparedListener

As a final example of working with video based media, the activity will now be extended further to demonstrate the mechanism for configuring a listener. In this case, a listener will be implemented that is intended to output the duration of the video as a message in the Android Studio Logcat panel. The listener will also configure video playback to loop continuously:

```
package com.ebookfrenzy.videoplayer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;
import android.widget.MediaController;
import android.util.Log;
import android.media.MediaPlayer;
```

```
public class VideoPlayerActivity extends AppCompatActivity {

    private VideoView videoView;
    private MediaController mediaController;
    String TAG = "VideoPlayer";

    private void configureVideoView() {

        final VideoView videoView =
                findViewById(R.id.videoView1);

        videoView.setVideoPath(
                "http://www.ebookfrenzy.com/android_book/movie.mp4");

        MediaController mediaController = new
                MediaController(this);
        mediaController.setAnchorView(videoView);
        videoView.setMediaController(mediaController);

        videoView.setOnPreparedListener(new
                    MediaPlayer.OnPreparedListener() {
                        @Override
                        public void onPrepared(MediaPlayer mp) {
                            mp.setLooping(true);
                            Log.i(TAG, "Duration = " +
                                    videoView.getDuration());
                        }
                    });
        videoView.start();
    }
}
```

Now just before the video playback begins, a message will appear in the Android Studio Logcat panel that reads along the lines of the following and the video will restart after playback ends:

```
11-05 10:27:52.256 12542-
12542/com.ebookfrenzy.videoplayer I/VideoPlayer: Duration = 6874
```

62.9 Summary

Tablet based Android devices make excellent platforms for the delivery of content to users, particularly in the form of video media. As outlined in this chapter, the Android SDK provides two classes, namely VideoView and

MediaController, which combine to make the integration of video playback into Android applications quick and easy, often involving just a few lines of Java code.