64. An Android Picture-in-Picture Tutorial

Following on from the previous chapters, this chapter will take the existing VideoPlayer project and enhance it to add Picture-in-Picture support, including detecting PiP mode changes and the addition of a PiP action designed to display information about the currently running video.

64.1 Changing the Minimum SDK Setting

Picture-in-Picture support is only available on Android 8 API 26 or later. When the VideoPlayer project was originally created, it was configured with a minimum SDK of Android 4.0 API 14 which will prevent the PiP code added in this chapter from compiling. To modify the SDK setting, locate the *Gradle Scripts –> build.gradle (Module: app)* file and increase the *minSdkVersion* setting from 14 to 26:

```
pply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    buildToolsVersion "26.0.0"
    defaultConfig {
        applicationId "com.ebookfrenzy.videoplayer"
        minSdkVersion 26
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
```

Once the change has been made, click on the *Sync Now* link in the yellow warning bar located across the top of the code editor.

64.2 Adding Picture-in-Picture Support to the Manifest

The first step in adding PiP support to an Android app project is to enable it within the project Manifest file. Open the *manifests -> AndroidManifest.xml* file and modify the activity element to enable PiP support:

```
.
.
<activity android:name=".VideoPlayerActivity"
    android:supportsPictureInPicture="true"
```

```
    android:configChanges="screenSize|smallestScreenSize|
            screenLayout|orientation">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
.
.
```

64.3 Adding a Picture-in-Picture Button

As currently designed, the layout for the VideoPlayer activity consists solely of a VideoView instance. A button will now be added to the layout for the purpose of switching into PiP mode. Load the *activity_video_player.xml* file into the layout editor and drag a Button object from the palette onto the layout so that it is positioned as shown in Figure 64-1:

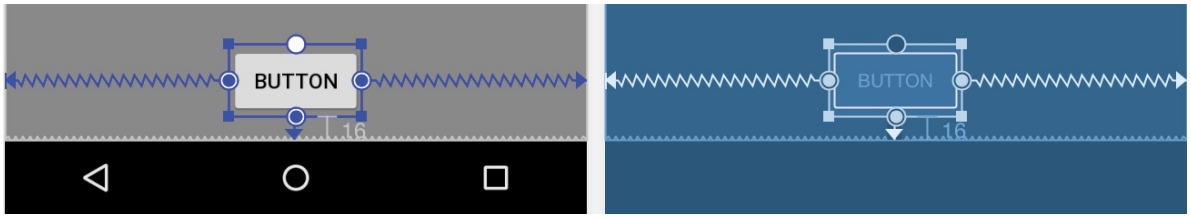

Figure 64-1

Change the text on the button so that it reads “Enter PiP Mode” and extract the string to a resource named *enter_pip_mode*. Before moving on to the next step, change the ID of the button to *pipButton* and configure the button to call a method named *enterPipMode*.

64.4 Entering Picture-in-Picture Mode

The *enterPipMode* onClick callback method now needs to be added to the *VideoPlayerActivity.java* class file. Locate this file, open it in the code editor and add this method as follows:

```
.
.
import android.view.View;
import android.app.PictureInPictureParams;
import android.util.Rational;
import android.widget.Button;
.
.
```

```
public void enterPipMode(View view) {

    Button pipButton = (Button) findViewById(R.id.pipButton);

    Rational rational = new Rational(videoView.getWidth(),
            videoView.getHeight());

    PictureInPictureParams params =
            new PictureInPictureParams.Builder()
                    .setAspectRatio(rational)
                    .build();

    pipButton.setVisibility(View.INVISIBLE);
    videoView.setMediaController(null);
    enterPictureInPictureMode(params);
}
```

The method begins by obtaining a reference to the Button view, then creates a Rational object containing the width and height of the VideoView. A set of Picture-in-Picture parameters is then created using the PictureInPictureParams Builder, passing through the Rational object as the aspect ratio for the video playback. Since the button does not need to be visible while the video is in PiP mode it is made invisible. The video playback controls are also hidden from view so that the video view will be unobstructed while in PiP mode.

Compile and run the app on a device or emulator running Android 8 and wait for video playback to begin before clicking on the PiP mode button. The video playback should minimize and appear in the PiP window as shown in Figure 64-2:

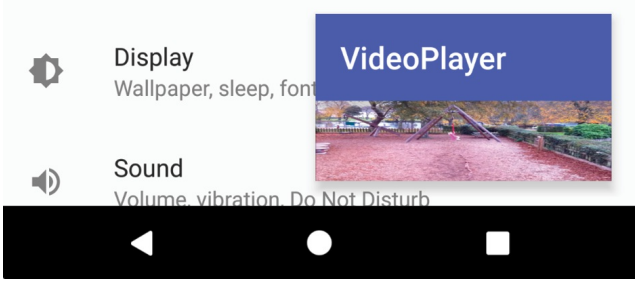

Figure 64-2

Although the video is now playing the PiP window, much of the view is obscured by the standard Android action bar. To remove this requires a change to the application theme style of the activity. Within Android Studio,

locate and edit the *app -> res -> styles.xml* file and modify the AppTheme element to use the *NoActionBar* theme:

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Compile and run the app, place the video playback into PiP mode and note that the action bar no longer appears in the window:

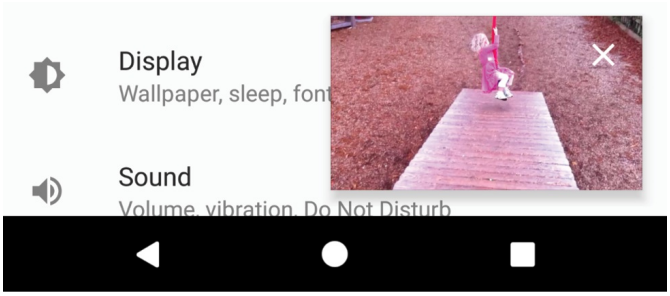

Figure 64-3

Click in the PiP window so that it increases in size, then click within the full screen mode markers that appear in the center of the window. Although the activity returns to full screen mode, note the button and media playback controls remain hidden.

Clearly some code needs to be added to the project to detect when PiP mode changes take place within the activity.

64.5 Detecting Picture-in-Picture Mode Changes

As discussed in the previous chapter, PiP mode changes are detected by overriding the *onPictureInPictureModeChanged()* method within the affected activity. In this case, the method needs to be written such that it can detect whether the activity is entering or exiting PiP mode and to take appropriate action to re-activate the PiP button and the playback controls. Remaining within the *VideoPlayerActivity.java* file, add this method now:

```
@Override
public void onPictureInPictureModeChanged(boolean
```

```
isInPictureInPictureMode) {
    super.onPictureInPictureModeChanged(isInPictureInPictureMode);

    Button pipButton = (Button) findViewById(R.id.pipButton);

    if (isInPictureInPictureMode) {

    } else {
        pipButton.setVisibility(View.VISIBLE);
        videoView.setMediaController(mediaController);
    }
}
```

When the method is called, it is passed a Boolean value indicating whether the activity is now in PiP mode. The code in the above method simply checks this value to decide whether to show the PiP button and to re-activate the playback controls.

64.6 Adding a Broadcast Receiver

The final step in the project is to add an action to the PiP window. The purpose of this action is to display a Toast message containing the name of the currently playing video. This will require some communication between the PiP window and the activity. One of the simplest ways to achieve this is to implement a broadcast receiver within the activity, and the use of a pending intent to broadcast a message from the PiP window to the activity. These steps will need to be performed each time the activity enters PiP mode so code will need to be added to the *onPictureInPictureModeChanged()* method. Locate this method now and begin by adding some code to create an intent filter and initialize the broadcast receiver:

```
.
.
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.widget.Toast;
.
.
public class VideoPlayerActivity extends AppCompatActivity {

    private BroadcastReceiver receiver;
```

```
    private VideoView videoView;
    private MediaController mediaController;
    String TAG = "VideoPlayer";
.
.
@Override
    public void onPictureInPictureModeChanged(
                    boolean isInPictureInPictureMode) {
        super.onPictureInPictureModeChanged(isInPictureInPictureMode);

        Button pipButton = (Button) findViewById(R.id.pipButton);

        if (isInPictureInPictureMode) {
            IntentFilter filter = new IntentFilter();
            filter.addAction(
              "com.ebookfrenzy.videoplayer.VIDEO_INFO");

            receiver = new BroadcastReceiver() {
                @Override
                public void onReceive(Context context,
                                      Intent intent) {
                    Toast.makeText(context,
                     "Favorite Home Movie Clips",
                            Toast.LENGTH_LONG).show();
                }
            };

            registerReceiver(receiver, filter);

        } else {
            pipButton.setVisibility(View.VISIBLE);
            videoView.setMediaController(mediaController);

            if (receiver != null) {
                unregisterReceiver(receiver);
            }
        }
    }
.
.
}
```

64.7 Adding the PiP Action

With the broadcast receiver implemented, the next step is to create a RemoteAction object configured with an image to represent the action within the PiP window. For the purposes of this example, an image icon file named *ic_info_24dp.xml* will be used. This file can be found in the *project_icons* folder of the source code download archive available from the following URL:

http://www.ebookfrenzy.com/retail/androidstudio30/index.php

Locate this icon file and copy and paste it into the *app -> res -> drawables* folder within the Project tool window:

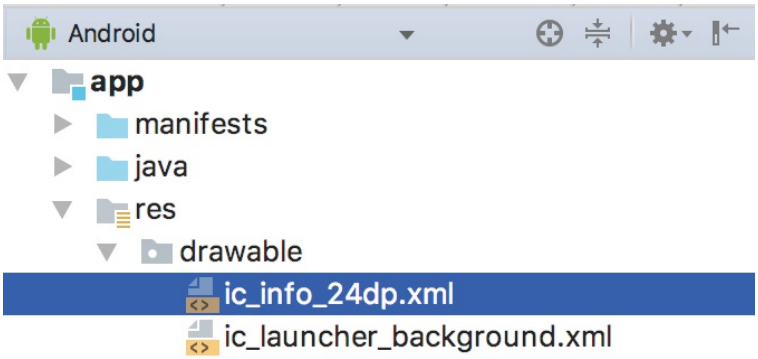

Figure 64-4

The next step is to create an Intent that will be sent to the broadcast receiver. This intent then needs to be wrapped up within a PendingIntent object, allowing the intent to be triggered later when the user taps the action button in the PiP window.

Edit the *VideoPlayerActivity.java* file to add a method to create the Intent and PendingIntent objects as follows:

```
.
.
import android.app.PendingIntent;
.
.
public class VideoPlayerActivity extends AppCompatActivity {

        private static final int REQUEST_CODE = 101;
.
.
.
        private void createPipAction() {
            Intent actionIntent =
                new Intent("com.ebookfrenzy.videoplayer.VIDEO_INFO");
```

```
            final PendingIntent pendingIntent =
                PendingIntent.getBroadcast(VideoPlayerActivity.this,
                  REQUEST_CODE, actionIntent, 0);
    }
.
.
}
```

Now that both the Intent object, and the PendingIntent instance in which it is contained have been created, a RemoteAction object needs to created containing the icon to appear in the PiP window, and the PendingIntent object. Remaining with the *createPipAction()* method, add this code as follows:

```
.
.
import android.app.RemoteAction;
import android.graphics.drawable.Icon;
.
.
private void createPipAction() {

    final ArrayList<RemoteAction> actions = new ArrayList<>();

    Intent actionIntent =
              new Intent("com.ebookfrenzy.videoplayer.VIDEO_INFO");

    final PendingIntent pendingIntent =
PendingIntent.getBroadcast(VideoPlayerActivity.this,
            REQUEST_CODE, actionIntent, 0);

    final Icon icon =
              Icon.createWithResource(VideoPlayerActivity.this,
                     R.drawable.ic_info_24dp);
    RemoteAction remoteAction = new RemoteAction(icon, "Info",
              "Video Info", pendingIntent);

    actions.add(remoteAction);
}
```

Now a PictureInPictureParams object containing the action needs to be created and the parameters applied so that the action appears within the PiP window:

```
private void createPipAction() {

    final ArrayList<RemoteAction> actions = new ArrayList<>();

    Intent actionIntent =
              new Intent("com.ebookfrenzy.videoplayer.VIDEO_INFO");

    final PendingIntent pendingIntent =
       PendingIntent.getBroadcast(VideoPlayerActivity.this,
            REQUEST_CODE, actionIntent, 0);

    final Icon icon =
              Icon.createWithResource(VideoPlayerActivity.this,
                     R.drawable.ic_info_24dp);
    RemoteAction remoteAction = new RemoteAction(icon, "Info",
              "Video Info", pendingIntent);

    actions.add(remoteAction);

    PictureInPictureParams params =
                new PictureInPictureParams.Builder()
                        .setActions(actions)
                        .build();

    setPictureInPictureParams(params);
}
```

The final task before testing the action is to make a call to the *createPipAction()* method when the activity enter PiP mode:

```
@Override
public void onPictureInPictureModeChanged(boolean
isInPictureInPictureMode) {
    super.onPictureInPictureModeChanged(isInPictureInPictureMode);
.
.
        registerReceiver(receiver, filter);
        createPipAction();
    } else {
        pipButton.setVisibility(View.VISIBLE);
        videoView.setMediaController(mediaController);
.
.
}
```

64.8 Testing the Picture-in-Picture Action

Build and run the app once again and place the activity into PiP mode. Tap on the PiP window so that the new action button appears as shown in Figure 64-5:

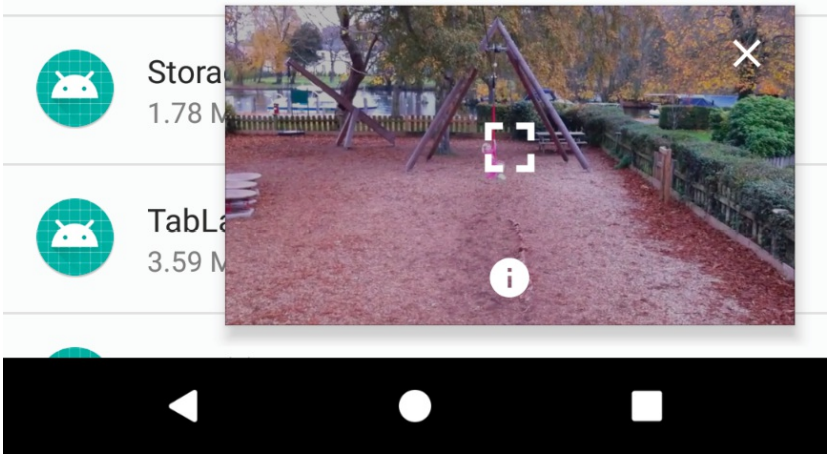

Figure 64-5

Click on the action button and wait for the Toast message to appear displaying the name of the video:

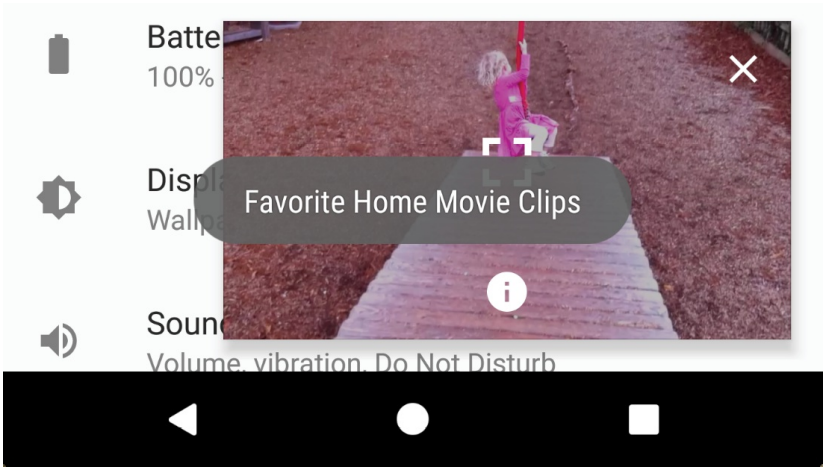

Figure 64-6

64.9 Summary

This chapter has demonstrated addition of Picture-in-Picture support to an Android Studio app project including enabling and entering PiP mode and the implementation of a PiP action. This included the use of a broadcast receiver and pending intents to implement communication between the PiP window and the activity.