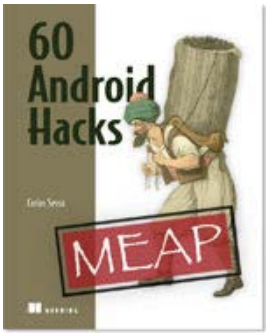
MANNING PUBLICATIONS



60 Android Hacks
By Carlos Sessa

Landscape is way cooler that portrait when it comes to viewing a video on your cell. To make the video full screen when the device is rotated, you need to tell the system that you'll handle the orientation changes yourself. In this article, based on a technique presented in 60 Android Hacks, the author shows you how to do that by changing the size and position of VideoView.

You may also be interested in…

VideoViews and Orientation Changes

Adding video to an application is a great way to create a richer user experience. I've seen applications that show information from a company and use fancy graphs done with videos. Sometimes videos are an easy way to present information in complex views without the need for coding the animation logic.

I noticed that when a video is available, users tend to turn the device to landscape to enjoy it, so in this hack I'll show you how to make the video full screen when the device is rotated.

To create this, we'll tell the system that we'll handle the orientation changes ourselves. When the device gets rotated, we'll change the size and position of the `VideoView`.

The first thing to do is create the layout we want for our `Activity`. For this hack I created a layout divided in two by a small line. The upper part will have a small bit of text on the left with a video on the right, and the bottom part will have a long description. While doing the XML for this view, instead of adding a `VideoView` where it should go, I added a `View` with a white background. This view will be used to copy its size and position to place the `videoView` correctly.

The `videoView` hangs from the root view at the same level as the portrait content. Placing the `videoView` there will allow us to change its size and position without needing to use two different layouts or changing the `videoView`'s parent when the rotation occurs. On the other hand, the white background view, called *portrait position*, is placed deeper in the tree.

Now that we have the layout, we can take care of the `Activity`'s code. The first thing to do is to enable ourselves to handle the orientation changes. To do this, we need to add `android:configChanges="orientation"` to the proper `<Activity>` element inside AndroidManifest.xml. Adding that attribute will cause the `onConfigurationChanged()` method to be called instead of restarting the `Activity` when the device is rotated.

When the orientation is changed, we need to change the video's size and position. For this, we'll call a private method called `setVideoViewPosition()`. Here's the content of this method:

```
private void setVideoViewPosition() {

  if (getResources().getConfiguration().orientation ==
    ActivityInfo.SCREEN_ORIENTATION_PORTRAIT) {                #1

    mPortraitContent.setVisibility(View.VISIBLE);              #2

    int[] locationArray = new int[2];
    mPortraitPosition.getLocationOnScreen(locationArray);      #3
```

```
      RelativeLayout.LayoutParams params =
        new RelativeLayout.LayoutParams(mPortraitPosition.getWidth(),
            mPortraitPosition.getHeight());

      params.leftMargin = locationArray[0];
      params.topMargin = locationArray[1];

      mVideoView.setLayoutParams(params);                    #4

    } else {

      mPortraitContent.setVisibility(View.GONE);             #5

      RelativeLayout.LayoutParams params =
        new RelativeLayout.LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.FILL_PARENT);

      params.addRule(RelativeLayout.CENTER_IN_PARENT);
      mVideoView.setLayoutParams(params);                    #6
    }
  }
```

As you can see in #1, the method `setVideoViewPosition()` is separated into two parts: the portrait and the landscape configurations. First, we make the portrait content visible in #2. Since the `videoView` will have the same position and size as the white view, in #3 we get its position set as the `videoView`'s layout parameters in #4.

Something similar is done in the second part, for landscape orientation. In this case, we first hide the portrait content in #5 and afterward we create the layout parameters to make the `videoView` use the whole screen. Finally, in #6 we set the layout parameters created to the `videoView`.

Summary

As I mentioned at the beginning of this hack, videos can be useful for improving your application content. You should know that the default `VideoView` class will respect the aspect ratio when resizing, and if you wish to make it fill the space available you'll need to override the `onMeasure()` method in your own custom view.

External links

Handling Runtime Changes: http://developer.android.com/guide/topics/resources/runtime-changes.html

Here are some other Manning titles you might be interested in:

Android in Action, Third Edition
W. Frank Ableson, Robi Sen, Chris King, and C. Enrique Ortiz

Android in Practice
Charlie Collins, Michael D. Galpin, and Matthias Kaeppler

Objective-C Fundamentals
Christopher K. Fairbairn, Johannes Fahrenkrug, and Collin Ruffenach

Last updated: January 27, 2012