

60 Android Hacks

Carlos Sessa



MEAP

 MANNING



MEAP Edition
Manning Early Access Program
60 Android Hacks version 2

Copyright 2012 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Layout

Hack 1 Centering views using weights

Hack 2 Avoid replication of XML views using the include tag

Hack 3 Lazy loading views with the ViewStub class

Hack 4 Customizing the chooser

Animation

Hack 5 Snappy transitions with TextSwitcher and ImageSwitcher

Hack 6 Adding eye candy to your ViewGroup's children

Hack 7 Handling lights-out mode

Hack 8 Doing animations over the canvas

Hack 9 TBD

View

Hack 10 Avoiding date validations with an EditText for dates

Hack 11 Removing the background to improve your activity startup time

Hack 12 Rounded borders for backgrounds

Hack 13 Getting the view's width and height in the onCreate() method

Hack 14 VideoViews and orientation changes

Hack 15 Formatting a TextView's text

Hack 16 Creating tabs with fragments

Hack 17 Adding glowing effects to texts

Hack 18 Toast's hacks

Hack 19 TBD

Tools

Hack 20 Using the Hierarchy Viewer tool to remove unnecessary views

Hack 21 Removing log statements before releasing

Hack 22 AOP in Android

Hack 23 Improving your logcat view

Hack 24 Collaborative internalization

Patterns

Hack 25 Creating fast adapters with a ViewHolder

Hack 26 MVP pattern in Android

Hack 27 Communicating with an Adapter using an Activity and a delegate

Hack 28 Architecture pattern with Android libraries

Hack 29 Making other apps depend on yours

Hack 30 Getting dependencies from the market

Hack 31 Offline mode

Service

Hack 32 Controlling Services with Alarms

Hack 33 TBD

AdapterView

Hack 34 Handling empty lists

Hack 35 Taking advantage of ListView's header

Hack 36 Handling orientation changes inside a ViewPager

Hack 37 Fast endless Lists

Hack 38 TBD

Third party

Hack 39 Balloon boxes in your Maps

Hack 40 Roboguice to the rescue

Hack 41 Empowering your application using Cocos2d-x

Boilerplate code

Hack 42 Getting user information when receiving feedback

Hack 43 Detected rooted devices

Hack 44 TBD

Fragmentation

Hack 45 Using android:targetSdkVersion to reach a larger audience

Hack 46 Using new APIs in older devices

Hack 47 Android 1.5 with different resource folders

Hack 48 TBD

Hack 49 TBD

Scala

Hack 50 Using Scala inside Android

Hack 51 Using Scala Actors in Android

Building tools

Hack 52 Handling dependencies with Apache Maven

Hack 53 Installing dependencies in a rooted device

Hack 54 Hudson magic for Android

Hack 55 Deploying to different devices using Apache Maven

BroadcastReceiver

Hack 56 BroadcastReceiver following Activity's lifecycle

Hack 57 Using BOOT_COMPLETE respecting others

User Experience

Hack 58 Replacing long press with quick actions

Hack 59 Creating a splash screen

Hack 60 Making your main screen a dashboard

Centering views using weights



1.5+

I was giving an Android talk to a group of developers, explaining how to create a view using an XML file, when someone asked: “What should I write if I want a button to be 50% of its parent width and centered?” At first I didn’t get what he was saying, but after he drew it on the board I understood. His idea is shown in figures 1.1 and 1.2.

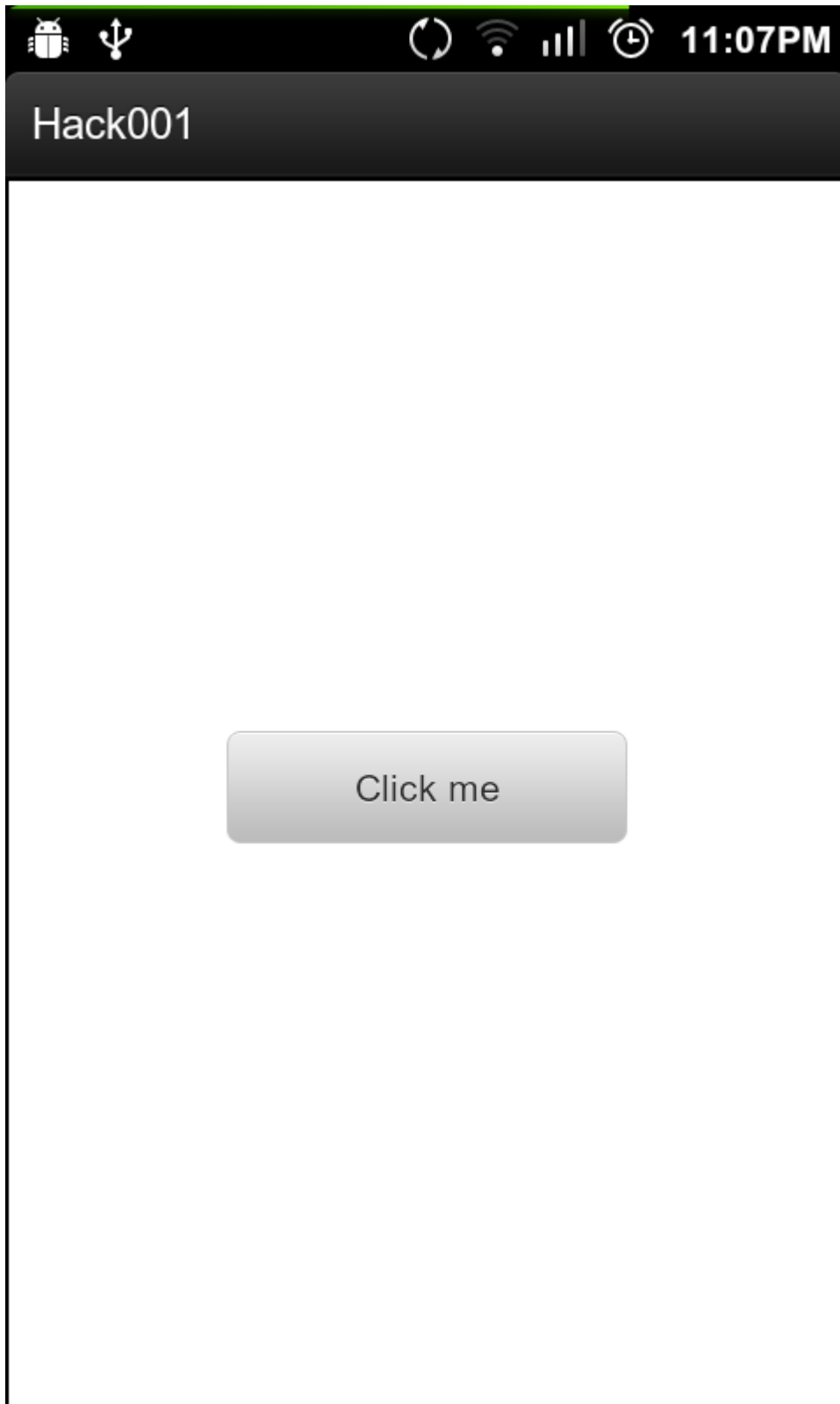


Figure 1.1 Button with 50% of its parent width (Portrait).

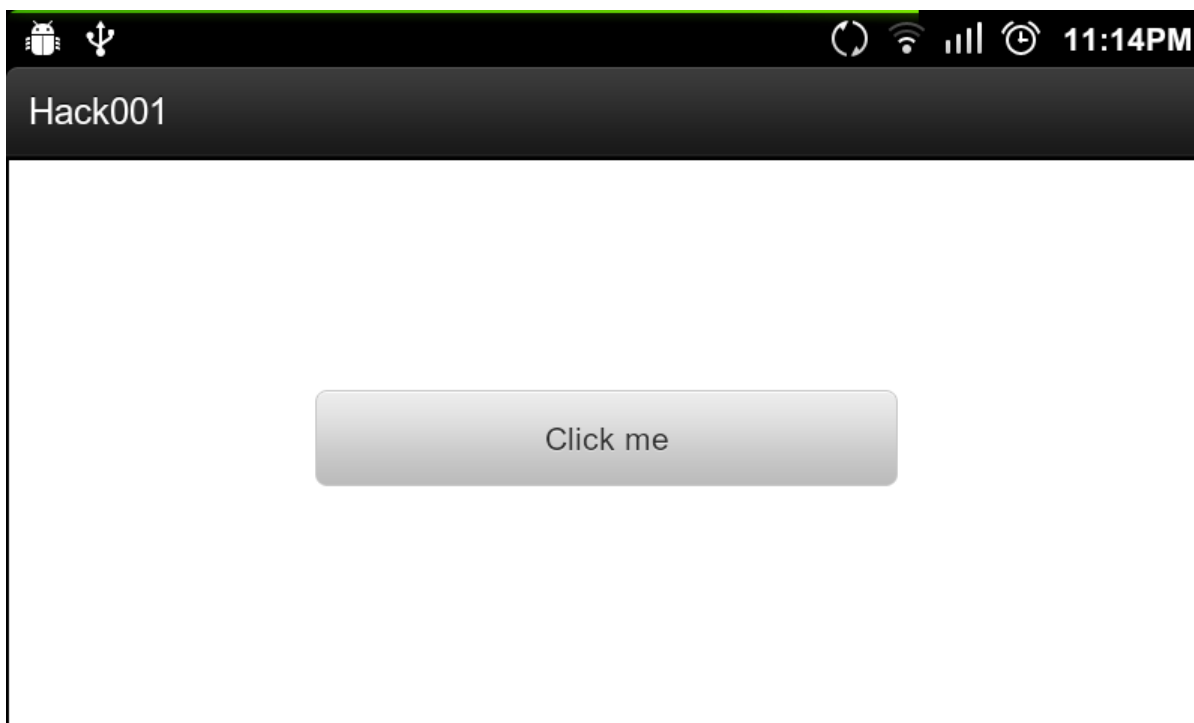


Figure 1.2 Button with 50% of its parent width (Landscape).

It looks simple, right? Now take five minutes to try to achieve it. In this chapter we'll look at how to solve this problem using the `LinearLayout`'s `android:weightSum` attribute in conjunction with `LinearLayout`'s children `android:layout_weight` attribute.

1.1 Combining `weightSum` and `layout_weight`

Android devices have different sizes, and as a developer you need to create XML in a way that should work for different screen sizes. Hard-coding sizes isn't an option, so we need something else to organize our views.

The `layout_weight` and the `weightSum` are two attributes which are used to fill up any remaining space inside the layout. The documentation for `android:weightSum` describes a scenario similar to what we're trying to achieve:

Defines the maximum weight sum. If unspecified, the sum is computed by adding the `layout_weight` of all of the children. This can be used for instance to give a single child 50% of the total available space by giving it a `layout_weight` of 0.5 and setting the `weightSum` to 1.0.

Imagine you need to place stuff inside a box. The percentage of available space would be the `weightSum` and the `layout_weight` would be the percentage

available for each item inside the box. For example, let's say the box has a `weightSum` of 1 and we have two items, A and B. A has a `layout_weight` of 0.25 and B has a `layout_weight` of 0.75. So item A will have 25% of the box space while B will get the remaining 75%.

The solution for our first situation we had is similar. We'll give the parent a certain `weightSum` and give the button half of that value as `android:layout_weight`. Here's the resulting XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFFFF"
    android:gravity="center"
    android:orientation="horizontal"
    android:weightSum="1"> ①

    <Button
        android:layout_width="0dp" ②
        android:layout_height="wrap_content"
        android:layout_weight="0.5" ③
        android:text="Click me"/>
</LinearLayout>
```

The `LinearLayout` reads the `android:weightSum` attribute ① and learns that the sum of the weights of its children needs to be 1. Its first and only child is the `Button` and since the button has its `android:layout_width` set to `0dp` ②, the `LinearLayout` knows that it must decide the button's width by the available space given by the `android:weightSum`. Since the `Button` has the `android:layout_weight` set to `0.5` ③, it will use exactly 50% of the available space.

A possible example would be a 200dp wide `LinearLayout` with its `android:weightSum` set to 1. The width of the `Button` would be calculated by doing:

```
Button's width + Button's weight * 200 / sum(weight)
```

Since the `Button's width` is `0dp`, the `Button's weight` is `0.5` and the `sum(weight)` was set to 1, the result would be:

```
0 + 0.5 * 200 / 1 = 100
```

1.2 The bottom line

Using `LinearLayout`'s `weight` is important when you want to distribute the available space depending on a percentage instead of using hard-coded sizes. If you're targeting Honeycomb and using Fragments, you'll notice that most of the examples place the different fragments in a layout using weights. Understanding how to use weights will add an important tool to your toolbox.

1.3 External links

- <http://developer.android.com/reference/android/widget/LinearLayout.html>

Index Terms

`layout_weight`
`weightSum`