Android: User Interface / Supporting Different Devices

https://developer.android.com/training/basics/supporting-devices/index.html

Ferruccio Damiani

Università di Torino
www.di.unito.it/~damiani

Mobile Device Programming
(Laurea Magistrale in Informatica, a.a. 2018-2019)

Outline

Outline

Three mains issues

- Supporting Different Languages and Cultures
  - How to support multiple languages and cultures with alternative string resources.
- Supporting Different Screens
  - How to optimize the user experience for different screen sizes and densities.
- Supporting Different Platform Versions
  - How to use APIs available in new versions of Android while continuing to support older versions of Android.

The following slides focus on "Supporting Different Screens":

- Although the system performs scaling and resizing to make your application work on different screens, you should make the effort to optimize your application for different screen sizes and densities.
- In doing so, you maximize the user experience for all devices and your users believe that your application was actually designed for their devices—rather than simply stretched to fit the screen on their devices.

Some guidelines

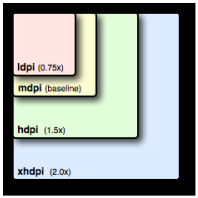
- Use wrap_content and match_parent
  - Use ConstraintLayout
- Use configuration qualifiers
  - Create a new directory in your project's res/ and name it using the format: <resources_name>-<qualifier>
    - <resources_name> is the standard resource name (such as drawable or layout)
    - <qualifier> is a configuration qualifier specifying the configuration for which these resources are to be used (such as hdpi or xlarge)

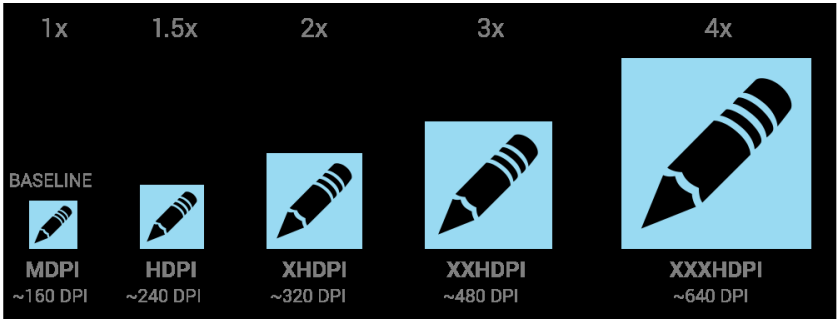
Configuration qualifiers that allow you to provide special resources for different screen configurations.

Screen characteristic	Qualifier	Description
Size	`small`	Resources for *small* size screens.
	`normal`	Resources for *normal* size screens. (This is the baseline size.)
	`large`	Resources for *large* size screens.
	`xlarge`	Resources for *extra-large* size screens.
Density	`ldpi`	Resources for low-density (*ldpi*) screens (~120dpi).
	`mdpi`	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)
	`hdpi`	Resources for high-density (*hdpi*) screens (~240dpi).
	`xhdpi`	Resources for extra-high-density (*xhdpi*) screens (~320dpi).
	`xxhdpi`	Resources for extra-extra-high-density (*xxhdpi*) screens (~480dpi).
	`xxxhdpi`	Resources for extra-extra-extra-high-density (*xxxhdpi*) uses (~640dpi). Use this for the launcher icon only, see note above.
	`nodpi`	Resources for all densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density.
	`tvdpi`	Resources for screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. If you find it necessary to provide tvdpi resources, you should size them at a factor of 1.33*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi.
Orientation	`land`	Resources for screens in the landscape orientation (wide aspect ratio).
	`port`	Resources for screens in the portrait orientation (tall aspect ratio).
Aspect ratio	`long`	Resources for screens that have a significantly taller or wider aspect ratio (when in portrait or landscape orientation, respectively) than the baseline screen configuration.
	`notlong`	Resources for use screens that have an aspect ratio that is similar to the baseline screen configuration.

Density[1].

- px = dp × (dpi / 160)
- Android says that Launcher icons on a mobile device must be 48 × 48 dp





[1] **px** = actual **pix**els on the screen. **dp** = **d**ensity-independent **p**ixel. **dpi** = **d**ots **p**er **in**ch

Alternative way (since Android 3.2)

Screen configuration	Qualifier values	Description
smallestWidth	sw<N>dp Examples: sw600dp sw720dp	The fundamental size of a screen, as indicated by the shortest dimension of the available screen area.
Available screen width	w<N>dp Examples: w720dp w1024dp	Specifies a minimum available width in dp units at which the resources should be used—defined by the <N> value
Available screen height	h<N>dp Examples: h720dp h1024dp	Specifies a minimum screen height in dp units at which the resources should be used—defined by the <N> value.

Examples of typical screen widths (smallestWidth):

- 320dp: typical phone screen
  - QVGA handset (240x320 ldpi)
  - hanset (320x480 mdpi)
  - high-density hanset (480x800 hdpi)
- 480dp: tablet/hanset (480x800 mdpi)
- 600dp: 7" tablet (600x1024 mdpi)
- 720dp: 10" tablet (720x1280 mdpi, 800x1280 mdpi)

Example [git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/KotlinDifferentScreen.git]

layout/activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <<android.support.constraint.ConstraintLayout
3      ...>
4      ...
5      <TextView
6          android:layout_width="wrap_content"
7          android:layout_height="wrap_content"
8          android:text="@string/hello"
9          android:id="@+id/textView1"
10         android:textStyle="bold"
11         android:textSize="20dp"
12         android:textColor="@color/colorText1"
13         ... />
14     ...
15 </<android.support.constraint.ConstraintLayout>
```

values-normal/strings.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="hello">Hello Normal World! </string>
4  </resources>
```

values-xlarge/strings.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="hello">Hello Extra Large World! </string>
4  </resources>
```

Different options

- Each layout can also be defined in an XML file in the res/layout
  - Layout aliases are then used to assign them to each configuration

File res/values/layouts.xml

```
<resources>
      <item name="main_layout" type="layout">
                              @layout/onepane_with_bar</item>
      <bool name="has_two_panes">false</bool>
</resources>
```

File res/values-sw600dp-land/layouts.xml

```
<resources>
      <item name="main_layout" type="layout">
                               @layout/twopanes</item>
      <bool name="has_two_panes">true</bool>
</resources>
```

FragmentActivity is a special activity to handle fragments

```
class MyActivity : FragmentActivity() {
    var mIsDualPane: Boolean = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.first)

        val secondView: View? = findViewById(R.id.second)
        mIsDualPane = secondView?.visibility === View.VISIBLE
    }
}
```

Manifest and devices

We must also declare in the manifest file which screens your application supports

- Through <supports-screens> manifest element
- If your application supports all screen sizes supported by Android (as small as 426dp $\times$ 320dp), then you don't need to declare this attribute, because the smallest width your application requires is the smallest possible on any device

Bitmaps

The system uses any size- or density-specific resources from your application and displays them without scaling

- If resources are not available in the correct density, the system loads the default resources and scales them up or down as needed

The system assumes that default resources (those from a directory without configuration qualifiers) are designed for the baseline screen density (mdpi)

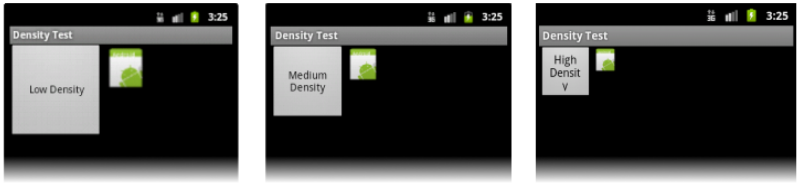
- A bitmap designed at 50×50 pixels for an mdpi screen is scaled to 75×75 pixels on an hdpi screen (if there is no alternative resource for hdpi)

Density independence

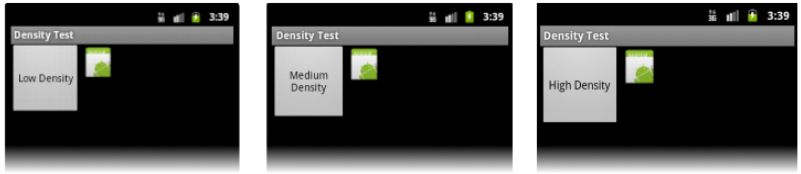
An application achieves "density independence" when it preserves the physical size (from the user's point of view) of user interface elements when displayed on screens with different densities.
The Android system helps your application achieve density independence in two ways:

- The system scales dp units as appropriate for the current screen density
- The system scales drawable resources to the appropriate size, based on the current screen density, if necessary

Example application without support for different densities, as shown on low, medium, and high-density screens.

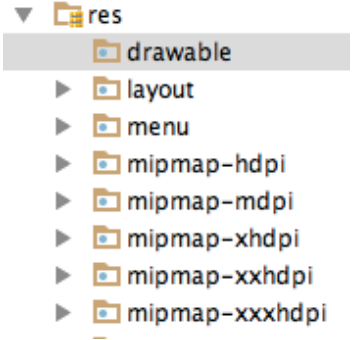


Example application with good support for different densities (it's density independent), as shown on low, medium, and high density screens.



Guidelines

- Use wrap_content, match_parent, or the dp unit for layout dimensions
- Do not use hard-coded pixel values in your application code
- Do not use AbsoluteLayout (deprecated)
- Use size and density-specific resources



For example, the following application resource directories provide different layout designs for different screen sizes and different drawables. Use the mipmap/ folders for launcher icons.

```
res/layout/my_layout.xml              // layout for normal screen size ("default")
res/layout-large/my_layout.xml        // layout for large screen size
res/layout-xlarge/my_layout.xml       // layout for extra-large screen size
res/layout-xlarge-land/my_layout.xml  // layout for extra-large in landscape orientation

res/drawable-mdpi/graphic.png         // bitmap for medium-density
res/drawable-hdpi/graphic.png         // bitmap for high-density
res/drawable-xhdpi/graphic.png        // bitmap for extra-high-density
res/drawable-xxhdpi/graphic.png       // bitmap for extra-extra-high-density

res/mipmap-mdpi/my_icon.png          // launcher icon for medium-density
res/mipmap-hdpi/my_icon.png          // launcher icon for high-density
res/mipmap-xhdpi/my_icon.png         // launcher icon for extra-high-density
res/mipmap-xxhdpi/my_icon.png        // launcher icon for extra-extra-high-density
res/mipmap-xxxhdpi/my_icon.png       // launcher icon for extra-extra-extra-high-density
```