Mobile Application Development

Resources

Resources

Resources are non-code files which are integrated into the application at compile time.
Examples: layouts, images, strings, files, audio, animations, styles.

Resources allows flexibility

You should always externalize resources to allow for flexibility with:

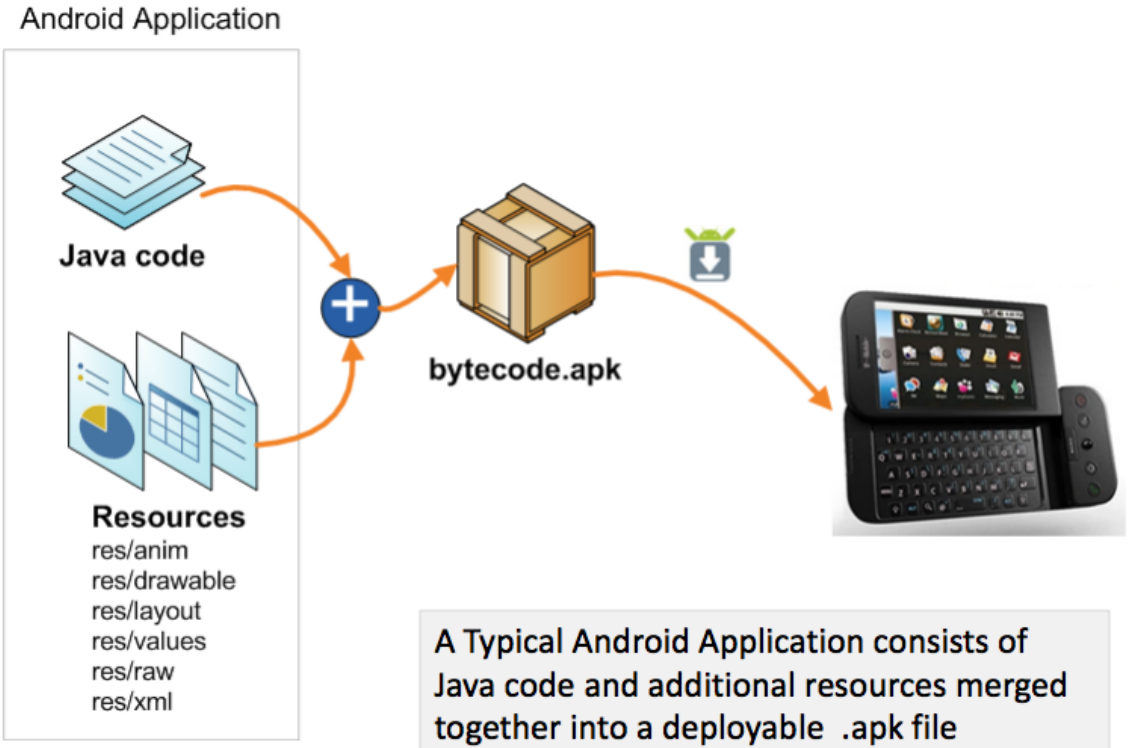
- Different form factors (phone, tablet)
- Different orientations (portrait, landscape)
- Different languages (e.g., English, Spanish)

Example: Layouts

Figure 1. Two different devices, each using the default layout (the app provides no alternative layouts).

Figure 2. Two different devices, each using a different layout provided for different screen sizes.

Resources

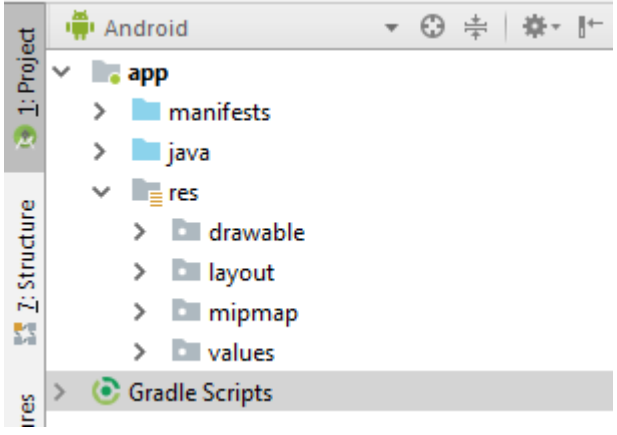


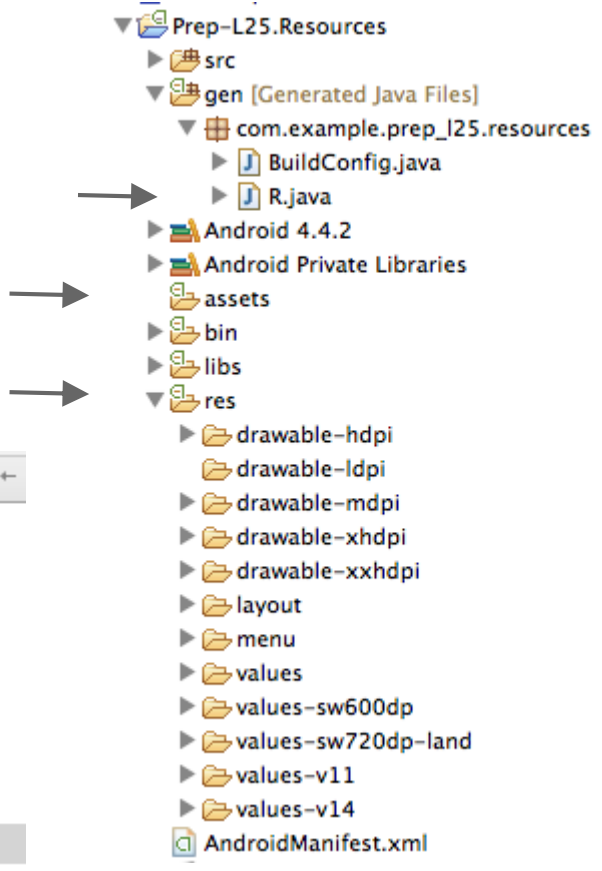
A Typical Android Application consists of Java code and additional resources merged together into a deployable .apk file

Resources

You place your resources under the appropriate /res or /assets subdirectory in your project's workspace.

Android creates a wrapper class, called **R**, that you can use to refer to these resources in your code.





Accessing Resources

- All items saved in the /res/ folder are indexed by ID. Those entries are stored in the `R.java` file.
- R.java is automatically generated by aapt in the build process.
- Never modify this file directly!
- Resources can be modified in code or XML

Accessing Resources in code

Syntax: R.[resource_type].[resource_name]

```
// Load a background for the current screen from a drawable resource
getWindow().setBackgroundDrawableResource(R.drawable.my_background_image) ;

// Set the Activity title by getting a string from the Resources object, because
//  this method requires a CharSequence rather than a resource ID
getWindow().setTitle(getResources().getText(R.string.main_title));

// Load a custom layout for the current screen
setContentView(R.layout.main_screen);

// Set a slide in animation by getting an Animation from the Resources object
mFlipper.setInAnimation(AnimationUtils.loadAnimation(this, R.anim.hyperspace_in));

// Set the text on a TextView object using a resource ID
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

Accessing Resources in XML

Syntax: <resource_type>/<resource_name>

```
<Button
    android:id="@+id/savemid“
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/submit" />
```

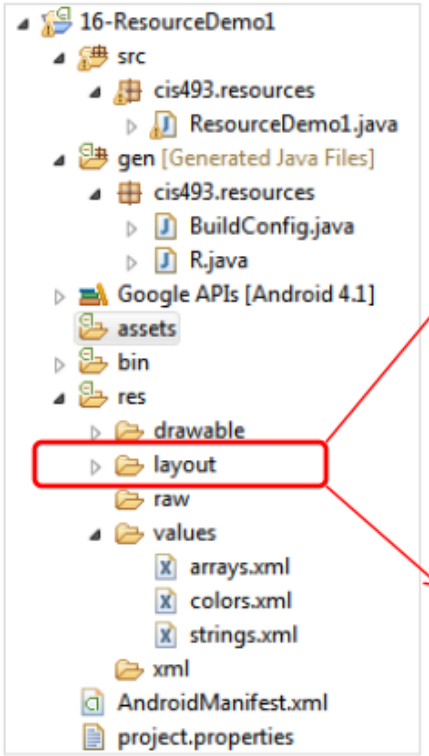
Resource Types

Resource Types

- Layouts
- Animation
- Drawables
- Strings
- Menus
- Styles
- Fonts
- And many more...

https://developer.android.com/guide/topics/resources/available-resources.html

Layout Resources




```
setContentView(R.layout.main);
```


```
setContentView(R.layout.screen2);
```

Animation Resources

Property Animation
Creates an animation by modifying an object's property values over a set period of time with an Animator.

Tween Animation
Creates an animation by performing a series of transformations on a single image with an Animation
Tween animations are saved in res/anim/ and accessed from the R.anim class.

XML file saved at `res/animator/property_animator.xml`:

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

Drawable Resources

Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into the following drawable resource subtypes:

- Bitmap files
- Nine-Patches (re-sizable bitmaps)
- State lists
- Shapes
- Animation drawables
- Etc.

Reference: http://developer.android.com/guide/topics/resources/drawable-resource.html

Creating from resource images (XML)

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/my_image"/>

<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/myninepatch"/>
```

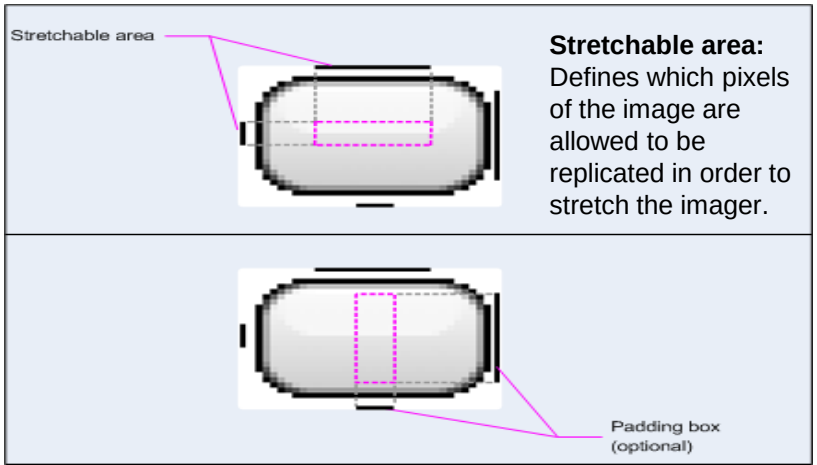
Creating from resource images (code)

```
ImageView i = new ImageView(this);
i.setImageResource(R.drawable.my_image);
```

Nine-patch drawable

Stretchable bitmap image that Android will automatically resize to accommodate the content

Example: standard Android buttons must stretch to accommodate strings of various lengths.



Stretchable area: Defines which pixels of the image are allowed to be replicated in order to stretch the imager.

Padding box (optional): Defines the relative area within the image that the contents of the View are allowed to lie within.

Shape Drawable

Allows you to draw primitive shapes:

- rectangle
- oval
- line
- ring
- etc

Shape Drawable Example

1. Create file drawable/gradient_box.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <padding android:left="10dp"
        android:top="10dp"
        android:right="10dp"
        android:bottom="10dp" />
    <corners android:radius="25dp" />
</shape>
```

1. Use as part of another element

```
  <TextView android:text="Beautiful"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:background="@drawable/gradient_box" />
```

MipMaps

- Generally used for icons to maintain their resolution
- Details:
  - Different home screen launcher apps on different devices show app launcher icons at various resolutions.
  - To avoid these display issues, apps should use the mipmap/resource folders for launcher icons. The Android system preserves these resources regardless of density stripping, and ensures that launcher apps can pick icons with the best resolution for display.

String Resources

In values/strings.xml (default language):

```
<string name="fox">fox</string>
```

In values-es/strings.xml (Spanish):

```
<string name="fox">zorro</string>
```

In code:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/fox" />
```

String Resources - Formatting

<b>, <i>, and <u> for bold, italics, and underlining

```
<string name="fox">
    <b>fox</b>
    <i>fox</i>
    <u>fox</u>
</string>
```

fox

fox

fox

Array Resources

Example: values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string-array name="colors">
   <item>red</item>
   <item>orange</item>
   <item>yellow</item>
   <item>green</item>
   <item>blue</item>
   <item>violet</item>
 </string-array>
</resources>
```

Example: Using an array for a spinner

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
ArrayAdapter<CharSequence> adapter =
ArrayAdapter.createFromResource(this,
    R.array.strings, android.R.layout.simple_spinner_item);
spinner.setAdapter(adapter);
```

Style Resource

A style resource defines the format and look for a UI.

A style can be applied to an individual View or to an entire Activity.

Similar to CSS - allows you to separate design from content.

Style Resource - Example

```
<!-- In values/styles.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

Style Resource - Example

```
<EditText
    style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

Style Resource - Themes

A **theme** is a style applied to an entire Activity or application

Every View in the Activity or application will apply each style property that it supports.

Set via android:theme attribute to the <activity> or <application> element in the Android manifest

- Can also set by setTheme method, but must be done before setContentView is called.

Resource directories supported inside project res/ directory

Directory	Resource Type
animator/	XML files that define property animations.
anim/	XML files that define tween animations. (Property animations can also be saved in this directory, but the animator/ directory is preferred for property animations to distinguish between the two types.)
color/	XML files that define a state list of colors. See Color State List Resource
drawable/	Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into the following drawable resource subtypes: •Bitmap files •Nine-Patches (re-sizable bitmaps) •State lists •Shapes •Animation drawables •Other drawables See Drawable Resources.
mipmap/	Drawable files for different launcher icon densities. For more information on managing launcher icons with mipmap/ folders, seeManaging Projects Overview.
layout/	XML files that define a user interface layout. See Layout Resource.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource.
raw/	Arbitrary files to save in their raw form. To open these resources with a raw InputStream, call Resources.openRawResource() with the resource ID, which is R.raw.*filename*. However, if you need access to original file names and file hierarchy, you might consider saving some resources in the assets/directory (instead of res/raw/). Files in assets/ are not given a resource ID, so you can read them only using AssetManager.
values/	•arrays.xml for resource arrays (typed arrays). •colors.xml for color values •dimens.xml for dimension values. •strings.xml for string values. •styles.xml for styles. See String Resources, Style Resource, and More Resource Types.
xml/	Arbitrary XML files that can be read at runtime by calling Resources.getXML(). Various XML configuration files must be saved here, such as a searchable configuration.

Supporting Multiple Screens by several configuration qualifiers

Concepts

Screen size - actual physical size, measured as the screen's diagonal.

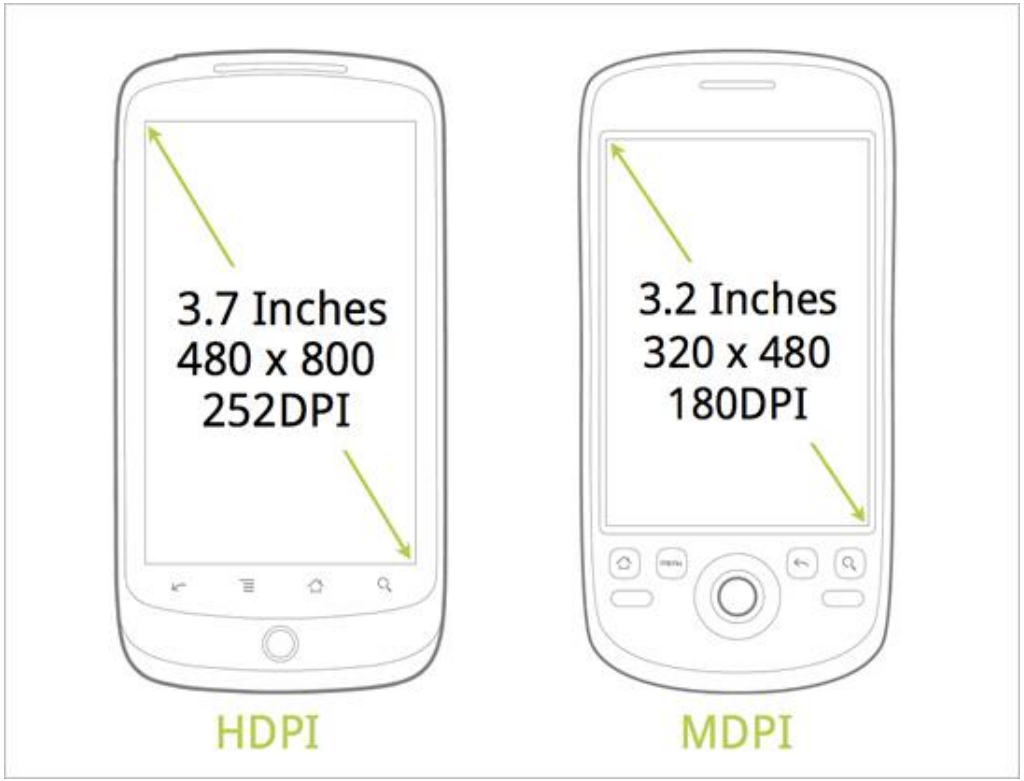
Screen pixel density - quantity of pixels within a physical area of the screen (dots per inch)

Resolution - total number of physical pixels on a screen

Screen orientation - port: Device is in portrait orientation (vertical)

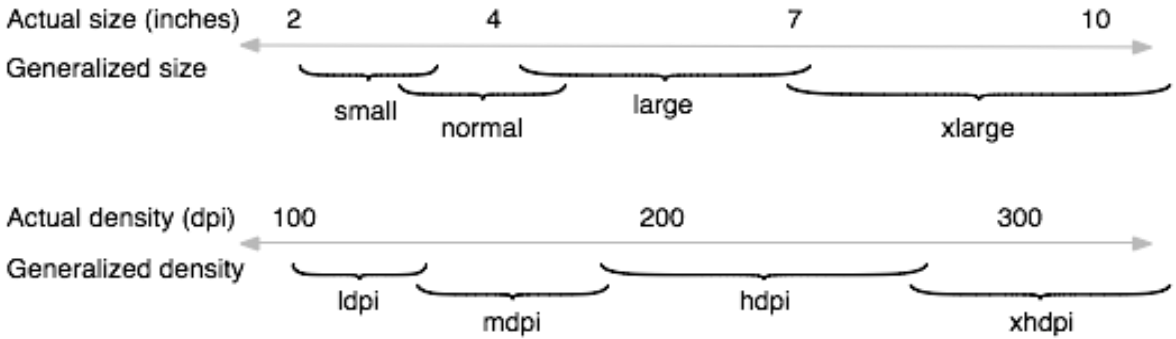
land: Device is in landscape orientation (horizontal)

Example

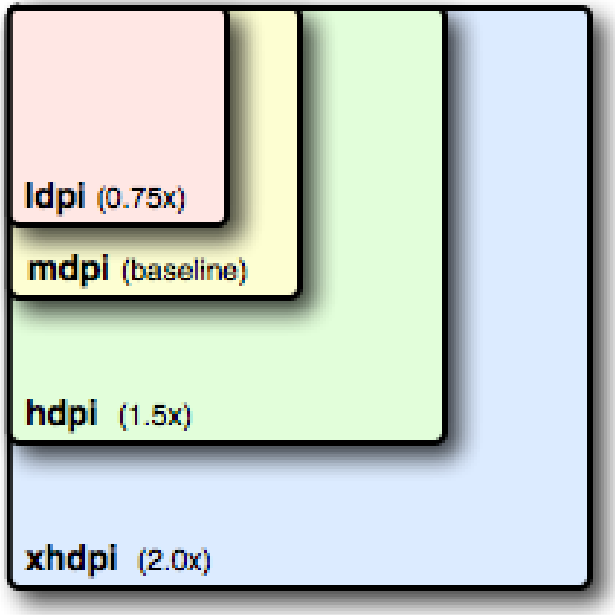

3.7 Inches
480 x 800
252DPI
3.2 Inches
320 x 480
180DPI
HDPI
MDPI

Value buckets

Screen size: small, normal, large, xlarge
Density: ldpi, mdpi, hdpi, xhdpi

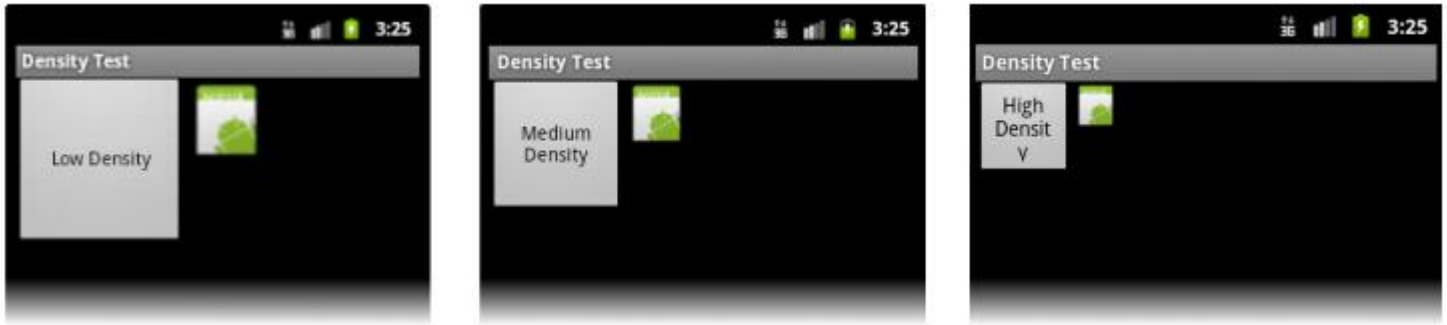


Relative size of densities

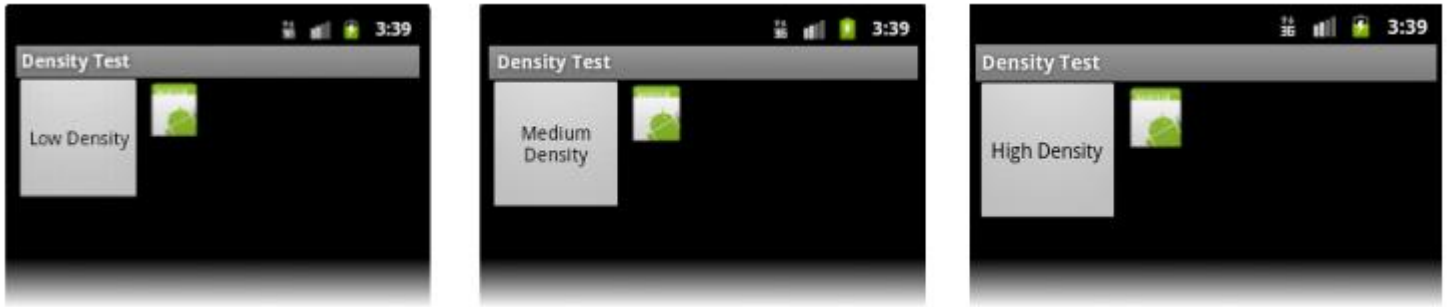


Density independence

Example app without support for different densities:



Example app with good support for different densities. **Layout is density independent.**



Units for Dimensions

Dimension	Description	Units / Physical Inch	Density Independent
px	Pixels	Varies	No
in	Inches	1	Yes
mm	Millimeters	25.4	Yes
pt	Points	72	Yes
dp	Density independent pixels	~160	Yes
sp	Scale independent pixels	~160	Yes

Always prefer dp or sp!

How to Support Multiple Screens

- Provide different layouts for different screen sizes
- Provide different bitmap drawables for different screen densities
- Always use dp or sp (for fonts) when specifying dimensions

Resource qualifiers

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).
	`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).

https://developer.android.com/guide/topics/resources/providing-resources.html